PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30, 17/40 — A1

(11) International Publication Number: WO 95/06292

(43) International Publication Date: 2 March 1995 (02.03.95)

(21) International Application Number: PCT/US94/09658

(22) International Filing Date: 24 August 1994 (24.08.94)

(30) Priority Data:
08/112,852 25 August 1993 (25.08.93) US

(71) Applicant: ASYMETRIX CORPORATION [US/US]; Suite 700, 110 - 110th Avenue, N.E., Bellevue, WA 98004-5840 (US).

(72) Inventors: HARDING, James, Allan; 3516 - 234th Avenue, S.E., Issaquah, WA 98027 (US). McCORMACK, Jonathan, Ian; 7661 Coal Creek Parkway, S.E., Renton, WA 98059 (US).

(74) Agents: LaRIVIERE, F., David et al.; P.O. Box 3140, Monterey, CA 93942 (US).

(81) Designated States: AM, AT, AU, BB, BG, BR, BY, CA, CH, CN, CZ, DE, DK, ES, FI, GB, GE, HU, JP, KE, KG, KP, KR, KZ, LK, LT, LU, LV, MD, MG, MN, MW, NL, NO, NZ, PL, PT, RO, RU, SD, SE, SI, SK, TJ, TT, UA, UZ, VN, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG), ARIPO patent (KE, MW, SD).

Published

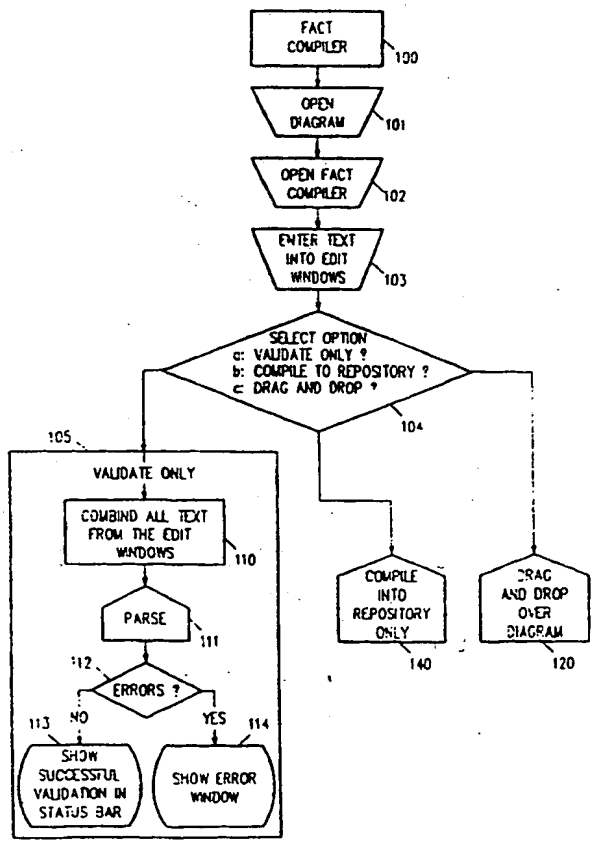
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND APPARATUS FOR THE MODELING AND QUERY OF DATABASE STRUCTURES USING NATURAL LANGUAGE-LIKE CONSTRUCTS

(57) Abstract

Computerized tools for modeling database designs and specifying queries of the data contained therein. Once it is determined that an information system needs to be created, the Fact Complier (100) of the present invention is invoked to create it. After creating the information system, the user creates a fact-tree, using the Fact_Tree_Specification Module (300), as a prelude to generating queries to the system. After creating the fact-tree, the user verifies that it is correct, using the Tree Interpreter, invoked as Fact_Tree_to_Description Module (500), of the present invention. Once the fact-tree has been verified, the Query Mapper of the present invention, invoked as Fact_Tree_to_SQL_Query Module (400), is used to generate information system queries.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgystan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SI	Slovenia
CI	Côte d'Ivoire	KZ	Kazakhstan	SK	Slovakia
CM	Cameroon	LI	Liechtenstein	SN	Senegal
CN	China	LK	Sri Lanka	TD	Chad
CS	Czechoslovakia	LU	Luxembourg	TG	Togo
CZ	Czech Republic	LV	Latvia	TJ	Tajikistan
DE	Germany	MC	Monaco	TT	Trinidad and Tobago
DK	Denmark	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	US	United States of America
FI	Finland	ML	Mali	UZ	Uzbekistan
FR	France	MN	Mongolia	VN	Viet Nam
GA	Gabon				

PATENT APPLICATION

METHOD AND APPARATUS FOR THE MODELING AND QUERY OF DATABASE STRUCTURES USING NATURAL LANGUAGE-LIKE CONSTRUCTS

TECHNICAL FIELD

This invention relates to the creation of computer database systems and the querying of data contained therein. In particular, the invention relates to computerized tools for modeling database designs and specifying queries of the data contained therein.

BACKGROUND ART

Computerized relational databases are used to form information systems which model real world issues and are composed of objects, the relationships i.e., facts, between those objects and the constraints and rules which govern these relationships and objects. Objects are physical or logical entities, capable of being uniquely identified. In this respect, objects are said to be essentially noun-like. Facts define the manner in which objects interact with one another, and are essentially verbs or are verb-like. Constraints modify or constrain the inter-relationships between objects and facts, and as such are analogous to adverbs and pronouns. As the use of information systems increases and the design of such systems advance, so increases the complexity of the real world issues they are expected to accurately model.

In creating an information system, a user needs to accurately transform the real world model, also known as the external view of the data, to its actual physical implementation, using a particular database language on a particular computer

system. This implementation is also called the physical view. In order to realize the power inherent in relational databases, it must be made possible for someone with no computing background or education to be able to design and implement information management systems and query meaningful data from them without having to learn a specific computer language.

The physical view of an information system is expressed in one of a number of database design languages. Examples of database design languages well known to those skilled in the art include Structured Query Language (SQL) and Microsoft Access. These database design languages are well adapted to carry out the storage and subsequent retrieval of data stored therein, but the languages themselves are both unnatural and highly technology specific. This means that database design languages are not typically used or understood by the end users of the information systems the languages model. The use of these design languages is a largely intuitive process practiced by database analysts who are familiar with the internal complexities of such languages.

The transformation of an information system from its external view to its physical view is time consuming, and at once formalized while remaining something of an art form. In order to assist database analysts in modeling data for information system design, several Computer Aided Software Engineering (CASE) tool sets have been developed, and are well known to those skilled in the art.

Prior art CASE tool sets were generally based upon entity-relationship modeling (ER). ER models, while providing a useful means of summarizing the main features of an application, are typically incapable of expressing many constraints and derivation rules that commonly occur in that application. An overview of ER-base tools may be found in Ovum (1992) and Reiner (1992). A state-of-the-art example is discussed in Czejdo et al. (IEEE Computer, Mar 1990, pp.26-37).

In order to capture much more of the detail of an application, object-role modelling (ORM), also known as fact-oriented modeling, was developed. Well known prior art versions of ORM include Natural-Language Information Analysis Method (NIAM), Binary-Relationship Modelling (BRM), Natural Object Role Modelling (NORM), and Predicator Set Model (PSM). One version of ORM,

Formal Object Role Modelling (FORM) is based on extensions to NIAM and has an associated language (FORML) with both graphical and textual forms (Halpin and Orlowska, 1992). FORM and FORML were developed in part by one of the inventors of the present invention.

The use of symbol-driven CASE tool sets provides a powerful instrument for conceptualizing the model of a given information system, but their use is not intuitively obvious to the untrained user. For such a user, being able to model information systems using a language with which the user is already facile is a more powerful approach. FORML provides the user with a natural language-like command set, and is thus readily learned.

Several CASE tool sets for object-role modeling exist. Among those known by persons skilled in the art are RIDL (Detroyer et al, 1988; Detroyer 1989; Nienhuys-Cheng 1990), GIST (Shoval et al, 1988) and IAST (Control Data, 1982). RIDL is currently marketed by Intellibase. These ORM-based CASE tool sets generally conform only to a binary-only version of ORM, although RIDL has recently added support for fact types of higher arity. In general, these systems are based upon the explicit "drawing" of symbols on diagram. Users of these tool sets typically specify their information systems by placing symbols directly on diagrams. In the typical CASE tool set, a different tool is used for each type of symbol used. The emphasis in these tool sets is on the notation of the symbols and what they mean, not the underlying semantics of the language upon which the notation rests.

An "optimal normal form" method for mapping from ORM to normalized relational tables was introduced in NIAM in the 1970's. This method ignored certain cases and provided a very incomplete specification of the methodology for constraint mapping. A significant extension of NIAM, capable of completely mapping any conceptual schema expressed in the graphic version of FORML to a redundancy-free, relational schema, was introduced as RMAP (Relational Mapping, Ritson and Halpin, 1992). RMAP differs from other mapping methods, such as RIDL-M, by enabling a wider variety of constraints; e.g., n-ary subset, equality, exclusion, closure and ring constraints.

Database professionals, using ORM-based CASE tool sets are markedly more productive than similar workers without them. A tool set which contains a mapping

schema such as RMAP is even more powerful, and results in further productivity. FORML based tool sets which implement RMAP represent the current state of the art with respect to ORM-based tool sets. Given FORML's graphical and textual language forms, the potential exists to combine the power, flexibility and precision of ORM based CASE tool sets with the ease and rapidity of use of graphical user interfaces common in modern computer systems. This will have the effect not only of further increasing the productivity of CASE tool sets in the hands of computer professionals, but will place these powerful software engineering tools in the hands of heretofore naive users as well.

While prior art natural language CASE tools do fulfill some of the promise of their basic concept, they lack the power of the symbol driven systems to model complex databases with facility. Until the present invention, there existed no CASE tool set for database design which combined the power, flexibility and accuracy of ORM using natural language-like constructs with a graphical user interface to translate the natural language-like constructs into ORM symbology and automatically map the conceptual schema so formed into a relational schema for implementation on a number of SQL-like database languages. The present invention effects a six-fold reduction in the number of user operations necessary to draw symbols on ORM-based diagrams by allowing users to type information in an approximately natural language. Users can think about the semantics of information and not waste time laboring on symbol drawing, which dampens the semantic thought process.

In addition to the ER and ORM-based prior art tool sets previously discussed, there have been efforts by other workers to automate the process of database specification using different methodologies. Some of the more pertinent attempts are described below.

U.S. Patent 4,688,196 to Thompson et. al. teaches a natural language interface generating system which allows a naive user to create and query a database based on a system of menu-driven interfaces. As the user addresses command words, in a natural language, to the interface generating system it provides a menu of words which could legally follow each word as it is input. The menu is provided by referencing pre-defined, resident files. Thompson calls these flies grammars and lexicons. The commands input by the user are translated by the system, which then

provides an automatic interactive system to generate the required interface in the following manner. After the database is loaded in, the interface generating system poses a series of questions to the user's technical expert. In response to these questions, the user or his expert must identify which tables in the database are to be used; which attributes of particular tables are key attributes; what the various connections are between the various tables in the database and what natural language connecting phrases will describe those relations.

U.S. Patent 4,939,689 to Davis et. al. teaches a system for the creation of database structures and subsequent querying of those structures by use of a text driven outliner system. The Davis system uses another form of resident dictionary table, which is again previously defined. In Davis, the user inputs a textual outline which defines the format of the database. This outline is then used to create data entry screens to facilitate data entry.

After creating database information systems (and assuming the data to populate those systems has been input), the information system must be accurately queried. Efforts by others skilled in the present art teach two broad strategies to enable the naive user to form queries.

The first prior art solution to the query generation problem is through the use of natural language parsers. This methodology takes a query which is input in a desired natural language such as English or Japanese, and parses the query into its component parts. Each component of the query is then used to form the translation of the original natural language query into a database language query. Until the present invention, this was typically accomplished by some form of resident database or dictionary file which translated the parsed command words and phrases into their respective equivalents in the database design language.

European Patent Application EP 0522591A2, filed 10 July, 1992 by Takanashi et. al., teaches a system typical of this "parse and look up" strategy, whereby a natural language query is entered and parsed into its constituent parts. The parser uses both a resident grammar table and a resident terminology dictionary to translate the meaning of individual command words and phrases into the database design language. The difficulty with fully implementing this solution is the richness and power i.e., the size and variable structure, of most natural languages. Each

possible word and many phrases must have a corresponding entry in the resident tables to make the system truly utile. If this is not done, the power of the natural language interface is substantially weakened in that a command will not be understood by the system.

The cost, both monetary and in computer overhead, of creating and maintaining a large, full-time resident natural language interface to any substantial information system is prohibitive. Furthermore, end users are still required to know the types of questions and keywords the parser and resident dictionary files will understand. This is because the resident table methodology does not fully account for the relationships between data objects and the constraints on those objects. For example, if a user wants to know Mr. Smith's age, it is not sufficient to ask *"How old is Smith ?"* since Smith might be a person or the Smith Tower. Instead the user must type *"How old is the person called Smith?"*. As a result, the learning curve for using natural language parsers is still extremely high.

The second solution to the query generation problem in the prior art is through the use of query tools. Query tools are based on the physical structures of the database and not the information contained therein. Information can be broadly categorized as a set of interacting conceptual objects, i.e. things you want to store - e.g., Person, Address, etc. Facts are relationships between objects - e.g. a Person *lives at an* address. When information is stored in a database, it is represented as a set of physical structures, e.g. tables. Absent considerable database expertise on the part of an end user, the physical representation of the data is invariably unintelligible to him or her. To enable, therefore, such a naive user to query data based on the physical structure it is stored in will require a significant training effort to ensure understanding of these physical structures.

In formulating a query using either a natural language parser or a physical structure query tool, one final issue remains. The user can never be sure that the query which is ultimately formed by either process is actually phrased correctly. When querying physical structures, absent significant training, the naive user doesn't understand the manner in which the data was stored. When using a natural language parser, the same problem arises due to the ambiguity inherent in that natural language. If, for instance, a user asked *"How old is Smith?"*, and the computer

answers "55", the answer may be for the person Smith, or the Smith Tower. This is reminiscent of the experience of a reporter who telegraphed Cary Grant's agent, asking about Mr. Grant's age. The reporter, sensitive to the cost per word of sending a telegraph, queried "HOW OLD CARY GRANT?". The actor, when the telegraph was inadvertently delivered to him, replied, again by telegraph, "OLD CARY GRANT JUST FINE". Clearly, unless the syntax of the query is correct, a naive user may retrieve an uncertain answer or an answer to an unintended query.

A common design feature of prior art CASE tools as previously discussed is the use of a pre-defined table or tables both to effect the translation of natural language inputs and to specify the exact nature of the data objects, facts and constraints as well as the interrelationships therebetween. As discussed, this methodology is costly, inefficient and not fully effective.

A further design feature of CASE tools currently in use for information system specification is their use of symbols instead of a natural language. A symbology-driven CASE tool set is at once imprecise and cumbersome, requiring several steps to perform the transformation from a chart of symbols to a database specification in a computer language.

There is therefore a need for apparatus that allows users to specify and create an information system using natural language or natural language-like commands, which will precisely specify the system's objects, facts and constraints without ambiguity or excessive overhead. This means should be capable of graphical depiction to define the interrelationships among the data elements in an unambiguous manner. The information used to create the system should be useable to define both the structure of the database itself as well as subsequent queries to that database once it is completed. There is a another need for a means for a naive user to be able to specify these queries to the system, again using natural language like commands which are not bound by previously entered definitions in a translation table. There is yet another need for a means for ensuring that any query which is created for the purpose of accessing the information system will, precisely and again without ambiguity, convey the user's intended question and return a correct, unambiguous answer.

DISCLOSURE OF INVENTION

The present invention provides a method and apparatus that allows users to:

1. Develop an information system description using a graphical user interface to a natural language-like computer language. One such language is FORML.

2. Specify the fact tree for query generation.

3. Check queries for semantic correctness.

4. Generate queries to the database system.

Once it has been determined that an information system needs to be created, the Fact Compiler of the present invention is invoked. The Compile function of the Fact Compiler enables a user to type in text, using a natural language-like computer language. One such language is FORML. The text is typed in a window provided by the system, and may contain objects (also referred to herein as nouns), facts (also referred to herein as fact types or sentences) and/or constraints. Using a translation function called "Drag and Drop over Diagram" and a graphical user interface, the user then drags the text from the entry window to the appropriate place over the ORM conceptual schema diagram of the Fact Compiler. The user then drops the text onto the diagram. The Fact Compiler validates the text entered and notifies the user of any errors encountered. During validation, the Fact Compiler first parses the text and creates an object list, a fact list and a constraint list in memory. Then the Fact Compiler iteratively compiles the text into the repository. The repository is essentially a "database of databases". Finally, the validated objects, facts and/or constraints are drawn in proper notation on the ORM conceptual schema diagram. At this point the information system specification may be considered complete.

After the information system has been created, the user may wish to check and/or edit the previously entered information. This is accomplished by using the Decompile function of the Fact Compiler. Decompile is essentially the reverse of the previously discussed Compile function, in that it takes an ORM conceptual schema diagram and returns a textual listing of the objects, facts and constraints entered in the repository. The user can use this listing to verify the information system specification or to edit the system as it exists.

Once the information system specification is complete, the conceptual schema depicted in the ORM representation of the information system is mapped to a

relational database using RMAP. The RMAP process is fully described in McCormack et al (1993), which is incorporated by reference as if fully set forth herein. By way of example, for an example set of facts:

Person lives at address

Person has Phone Number

Person studies Subject

Subject is taught by Person

if the relational database associated with an example fact tree is:

Person _Table: (Person, Address)

Phone _Table: (Person, Phone Number)

Studies _Table: (Person, Subject Studied)

Subject_Table: (Subject, Teacher Person)

The associated RMAP mappings would be:

FACT	TABLE	FIRST NOUN COLUMN	SECOND NOUN COLUMN
Person lives at address	Person_Table	Person	Address
Person has Phone Number	Phone_table	Person	Phone Number
Person studies Subject	Studies_Table	Person	Subject Studies
Subject is taught by Person	Subject_Table	Person	Teacher Person

The first step in query processing is specifying the fact-tree. In Fact-Tree Specification, the user selects a noun relevant to the query. For example, if the user wanted to find out the address, phone number, subjects studied, and teachers of Mr. Smith, they would start with the Person noun because the query is basically about a person. After choosing Person as the root of the query, they can select more information about the person - to find out their address etc. The only information they are able to select is the information contained in the facts about the person, i.e.

- A person lives at an address.
- A person has a phone number.
- A person studies a subject.
- A person teaches a subject.

This set of facts is all of the information possible about a particular person. The information is displayed conceptually and the user didn't need to know any special keywords or phrases. In this case the user would select the facts

- A person lives at an address.

0 A person has a phone number.

O A person studies a subject.

O A subject is taught by a person.

since that is what they want to know about Mr. Smith. This would build up the following fact-tree.

Person

_that lives at an address

_that has a phone number

_that studies a subject

_that is taught by a person.

Finally, the user would restrict the person at the root of the tree to be equal to Mr. Smith, since this is the only person they are interested in.

The meaning of the final tree is: Show the person Mr. Smith, the address that he lives at, the phone numbers that he has, the subject that he studies, and for the subjects he studies, show the people that teach those subject.

After generating the fact-tree, the user verifies that the fact-tree is correct using the Tree Interpreter of the present invention. Doing so will preclude the possibility of an ambiguous query being generated. In use, the Tree Interpreter algorithm constructs a natural language description of the fact-tree. This algorithm is a recursive depth-first search function which is described in the following best mode section. This interpretation allows the user to verify that the question he or she is asking will retrieve the information desired.

Once the user has specified the fact-tree and checked it using the Tree Interpreter, all that remains to do is generate the relational query itself. The algorithm to do this is again recursive on fact-tree nodes, and is detailed in the following section detailing the best mode of carrying out the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The various features and advantages of the present invention may be more readily understood with reference to the following detailed description of the preferred embodiments taken in conjunction with the accompanying drawings, wherein like reference characters designate like elements.

FIG. 1 is a diagram of the external view of a digital, programmable, general purpose computer configured for operation of the present invention.

FIG. 2 is a block diagram of the computer of FIG. 1 configured with the present invention.

FIG. 3 is a flow chart illustrating the initial selection menu of the present invention, after selecting the Fact Compiler of the present invention.

FIG. 4 is a flow chart illustrating the Fact Compiler of the present invention, including it's three main functions.

FIG. 5 is a flow chart illustrating the Drag and Drop over Diagram function of the Fact Compiler of the present invention.

FIG. 6 is a flow chart of the Drag and Drop Parse function invoked by the Drag and Drop over Diagram function.

FIG. 7 is a flow chart of the Compile into Repository Only function invoked by the Fact Compiler.

FIG. 8 is a flow chart of the Parse function invoked by the Compile into Repository Only function.

FIG. 9 is a flow chart of the Scan function invoked by the Parse function.

FIG. 10 is a table indicating the procedure for entering an error condition according to the present invention.

FIG. 11 is a flow chart illustrating the Look for Object Specifications function invoked by the Parse function.

FIG. 12 is a flow chart representing the Look for Fact Specifications function invoked by the Parse function.

FIG. 13 is a flow chart representing the Look for Constraint Specifications function invoked by the Parse function.

FIG. 14 is a flow chart illustrating the Allocate a New Object function invoked by the Look for Object Specifications function.

FIG. 15 is a flow chart illustrating the Allocate a New Fact function invoked by the Look for Fact Specifications function.

FIG. 16 is a flow chart illustrating the Allocate a New Constraint function invoked by the Look for Constraint Specifications function.

FIG. 17 is a flow chart representing the Update Record in Repository function invoked by both the Drag and Drop Parse and Compile into Repository Only functions.

FIG. 18 is a flow chart illustrating the initial selection menu of the present invention, after selecting Fact Tree Specification according to the present invention.

FIG. 19 is a flow chart illustrating the Fact Tree Formation function and selection of either the Query Mapper or Tree Interpreter functions of the present invention.

FIG. 20 is a flow chart representing the Fact Tree to SQL Query function invoked when a user selects the Query Mapper function of the present invention.

FIG. 21 is a flow chart illustrating the Node to SQL function invoked by the Fact Tree to SQL function.

FIG. 22 is a flow chart representing the Create Join function invoked by the Node to SQL function.

FIG. 23 is a flow chart illustrating the Add Selector 1 function invoked by the Node to SQL function.

FIG. 24 is a flow chart illustrating the Add Selector 2 function invoked by the Node to SQL function.

FIG. 25 is a flow chart of the Fact Tree to Description function invoked when a user selects the Tree Interpreter of the present invention.

FIG. 26 is a flow chart of the Create Text for Root function invoked by the Fact Tree to Description function.

FIG. 27 is a flow chart illustrating the Create Text for Node function invoked by the Fact Tree to Description function.

BEST MODE OF CARRYING OUT THE INVENTION

The preferred embodiment of the present invention incorporates computer system 1 configured as shown in Figure 1. Computer system 1 is a programmable digital computer. The invention is executable on an IBM compatible computer having an Intel 80386 or higher chip set, operating under the MS-DOS operating system, version 5.0 or higher. A minimum or 6 megabytes of available RAM is required for execution, as is a minimum of 6 megabytes of available hard disk storage space. These computers typically include a CPU, main storage, I/O

resources, and a user interface, including a manually operated keyboard and mouse. The present invention also requires a graphical user interface program: Microsoft Windows is one well known example.

The present invention was programmed on an IBM compatible computer having an Intel 80486 chip set, running Microsoft MS-DOS operating system, version 5.0. Microsoft Windows Version 3.1 was installed to provide the required graphical user interface. Finally, the system whose description follows was programmed in the Borland C language.

Figure 2 depicts the bus structure of the general purpose programmable computer of Figure 1, with the present invention implemented thereon.

Referring now to Figure 3, a user initiates the system at manual input 50 and selects the desired function at function selection 51.

The present invention provides a method and apparatus that allows users to:

1. Develop an information system description using a graphical user interface to a natural language-like computer language. One such language is FORML.

2. Specify the fact tree for query generation.

3. Check queries for semantic correctness.

4. Generate queries to the database system.

Once it has been determined that an information system needs to be created, the Fact Compiler of the present invention is invoked. The Compile function of the Fact Compiler enables a user to type in text, using a natural language-like computer language. One such language is FORML. The text is typed in a window provided by the system, and may contain objects, facts and/or constraints. Using a translation function called "Drag and Drop over Diagram" and a graphical user interface, the user then drags the text from the entry window to the appropriate place over the ORM conceptual schema diagram of the Fact Compiler. The user then drops the text onto the diagram. The Fact Compiler validates the text entered and notifies the user of any errors encountered. During validation, the Fact Compiler first parses the text and creates an object list, a fact list and a constraint list in memory. Then the Fact Compiler iteratively compiles the text into the repository. The repository is essentially a "database of databases". Finally, the validated objects, facts and/or constraints are drawn in proper notation on the ORM

conceptual schema diagram. At this point the information system specification may be considered complete.

After the information system has been created, the user may wish to check and/or edit the previously entered information. This is accomplished by using the Decompile function of the Fact Compiler. Decompile is essentially the reverse of the previously discussed Compile function, in that it takes an ORM conceptual schema diagram and returns a textual listing of the objects, facts and constraints entered in the repository. The user can use this listing to verify the information system specification or to edit the system as it exists.

Once the information system specification is complete, the conceptual schema depicted in the ORM representation of the information system is mapped to a relational database using RMAP. The RMAP process is fully described in McCormack et al (1993), which is incorporated by reference as if fully set forth herein. By way of example, for an example set of facts:

Person lives at address

Person has Phone Number

Person studies Subject

Subject is taught by Person

if the relational database associated with an example fact tree is:

Person _Table: (Person, Address)

Phone _Table: (Person, Phone Number)

Studies _Table: (Person, Subject Studied)

Subject_Table: (Subject, Teacher Person)

The associated RMAP mappings would be:

FACT	TABLE	FIRST NOUN COLUMN	SECOND NOUN COLUMN
Person lives at address	Person_Table	Person	Address
Person has Phone Number	Phone_table	Person	Phone Number
Person studies Subject	Studies_Table	Person	Subject Studies
Subject is taught by Person	Subject_Table	Person	Teacher Person. if the relational

database associated with an example fact tree is:

The first step in query processing is specifying the fact-tree. In Fact-Tree Specification, the user selects a noun relevant to the query. For example, if the user wanted to find out the address, phone number, subjects studied, and teachers of Mr. Smith, they would start with the Person noun because the query is basically about

a person. After choosing Person as the root of the query, they can select more information about the person - to find out their address etc. The only information they are able to select is the information contained in the facts about the person, i.e.

- A person lives at an address.
- A person has a phone number.
- A person studies a subject.
- A person teaches a subject.

This set of facts is all of the information possible about a particular person. The information is displayed conceptually and the user didn't need to know any special keywords or phrases. In this case the user would select the facts

- A person lives at an address.
- A person has a phone number.
- A person studies a subject.
  - A subject is taught by a person.

since that is what they want to know about Mr. Smith. This would build up the following fact-tree.

Person

_that lives at an address

_that has a phone number

_that studies a subject

_that is taught by a person.

Finally, the user would restrict the person at the root of the tree to be equal to Mr. Smith, since this is the only person they are interested in.

The meaning of the final tree is: Show the person Mr. Smith, the address that he lives at, the phone numbers that he has, the subject that he studies, and for the subjects he studies, show the people that teach those subject.

After generating the fact-tree, the user verifies that the fact-tree is correct using the Tree Interpreter of the present invention. Doing so will preclude the possibility of an ambiguous query being generated. In use, the Tree Interpreter algorithm constructs a natural language description of the fact-tree. This algorithm is a recursive depth-first search function which can be summarized as follows:

```
function: Interpret_Tree (fact-tree_node) // Interpret_Tree operates on a node of
          the fact-tree
    begin
      If the node is the root of the tree then
          noun is the noun in the node          e.g. Person
          Print 'For all noun(s)'               e.g. For all Person(s)
          if the node has a restriction then    e.g. is equal to Mr. Smith
            print "(where noun restriction)" e.g.(where Person = Mr.Smith)
          Print "show:" and move on to a new line
      otherwise
          noun is the noun in the node                          e.g. Address
          parent-noun is the noun in the node's parent          e.g. Person
          phrase is the phrase in the noun                      e.g. lives at
          Print "the noun(s) that the parent-noun phrase"       e.g. the Address(es)
                                                                   that the person
                                                                   lives at
          if the node has a restriction then                    e.g. is equal to
                                                                       Seattle
            print "(where noun restriction)"                    e.g. (where address
                                                                       = Seattle)
          if the node has any children then
            print ", and for those noun(s) show:"
          move on to a new line
      for all children of the node do
          call Interpret_Tree on the child-node
    end
The result of Interpret_Tree on the example fact-tree would be
    For all Person(s) (where Person = Mr. Smith) show:
          the Address that the Person lives at
          he Phone Number that the Person has
          the Subjects that the Person studies,
          and for those subjects show the Person(s) that the Subject is taught
          by.
```

This interpretation allows the user to verify that the question he or she is asking will retrieve the information desired.

Once the user has specified the fact-tree and checked it using the tree interpreter, all that remains to do is generate the relational query itself. The algorithm to do this is again recursive on fact-tree nodes.

function Create_Query (fact-tree_node)

begin

node is the node being mapped by this call to the function

child i ... childn are the children of *node*

sentence1, ... sentencen. are the respective sentences for *child i ... childn*

each sentence i, (i=1...n) has a mapping' associated with it. The mapping corresponds to the relational structure used to represent the sentence and contains the table that the sentence was mapped to, the column for the first noun and the column for the second noun. For example, the sentence Person *lives at Address* maps to the Person table, with noun1 (person) being column 1, and noun2 (address) being column 2.

Join all of the mappings for sentences 1..n together using outer joins based on the noun in *node* and the respective positions of that noun in sentences 1 to n. Form a query, including restriction when required.

An SQL query representative of this applied to the example fact-tree would be:

```
select Person. Person, Phone Number, Subject
from Person, Person has Phone_Number, Person_studies_Subject
Outer join Person.Person = Person has Phone_Number.Person
Outer join Person.Person = Person_studies_Subject.Person
 where Person.Person = 'Mr. Smith' ...
```

If any *child*i i=1..n have children, apply Create_Query to child and use an outer join to include the result into the existing query. In the example fact-tree, this would result in Create_Query being executed on the Subject node of the Person that studies Subject branch and would result in the query:

```
select  Person.Person, Phone Number, Person_studies_Subject. Subject
        Subject.Person
```

```
from Person, Person_has_Phone Number, Person_studies_Subject, Subject
Outer join Person.Person=Person_has_Phone_Number.Person
Outer join Person.Person = Person_studies_Subject.Person
Outer join Person_studies_Subject.Subject = Subject.Subject
 where Person.Person = 'Mr. Smith.....
```

Note that SQL is used as a notational convenience and its use has no bearing on the theory behind the algorithm. It is a particular feature of the present invention that any relational language could have been used. The advantages of the Fact Compiler, Query Mapper and Tree Interpreter algorithms of the present invention are that they substantially reduce the number of concepts and amount of training required for a naive end-user to express meaningful queries in a relational database. The algorithm set of the present invention allows users to form conceptual queries without having to know keywords and physical structures. Also, the algorithms of the present invention provide a generated natural language description of the query to assure that the query is correct in syntax. These advantages are illustrated by contrasting the previously described example with the same example expressed in the state of the art natural language and SQL implementations.

To express the query in natural language, the user would need to construct and type in the query:

Show me the person called Mr. Smith, his Address, his Phone Number, the subjects he studies, and for those subjects show people who teach them

The distinct pieces of information required to phrase this query are:

1. An overall knowledge of how to express the query (having to include keywords like person, etc.)
2. The knowledge that you could ask for Address, Phone Numbers, Subjects, etc ....

Fact Compiler

The fact compiler of the present invention is selected at 100. The fact tree of the present invention is selected at 300. As explained above, a Fact Compiler is provided by the present invention, a detailed description of which follows. Referring now to Figure 4, after selecting fact compiler 100, the user opens a diagram which represents one level of the information system to be modelled. After

opening the fact compiler diagram, the user types in a factual sentence, an object type, or a constraint, using a natural language-like computer language. One such language is FORML. An example of such an input is "the INSTRUCTOR with the ID "100" is qualified to teach the SUBJECT with the name "database design" at the SUBJECT LEVEL 300". At this point, the user may select one of three options: validate the input using Validate Only function 105; compile the information only into the repository 140; or to drag and drop the fact over the diagram at function 120.

Referring now to Validate Only function 105 of Figure 4 of the present invention, after all text has been combined from the edit windows at 110, it is parsed into its component words using function 111. An error checker at 112 determines if there are errors in the text which is input. If there are no errors, the system indicates successful validation at 113. If error checker 112 determines that there are errors in the textual input, the errors will be shown at error window 114.

Drag and Drop over Diagram function 120, is detailed in Figure 5. After the user selects Drag and Drop over Diagram at 120, the user utilizing a mouse moves a pointer over the icon from the edit window at 121. The user presses the left mouse button down and holds it down at 122. The system will test for the type of item which was input in the edit window at 123. Item kind selector 124 will change the nature of the cursor depending upon the type of information input in the edit window. If the data input is a fact, the cursor will change to a fact cursor at 125. If the data input is a constraint, the cursor will be changed to a constraint cursor at 126. If the data input is an object, the cursor will change to an object cursor at 127. The user drags the modified cursor over the diagram at 128. The edit window will disappear while the cursor is being moved. If the cursor moves over a non-drawing area the cursor change to a "Can't Draw" cursor. In the event the user needs to reorient the direction of the data which is input, the right mouse button is used. For each click of the right mouse button, the icon changes its orientation 90° at 130. When the user releases the left mouse button at 131, the cursor will drop the data over the diagram. Cursor checker 132 determines if the cursor is actually over the diagram or not. If it is, Drag and Drop Parse function 135 is invoked. If not, an indication is given the user that the cursor is not in the diagram.

Drag and Drop Parse function 135 is detailed in Figure 6. After collecting text at the current edit window, the text is parsed using Parse function 142. After parsing an error checker 161 determines if an error has been made in the textual input. If no error has been made, the record is updated to the repository using the Update Record to Repository function 149. After the record has been updated to the repository, the item is drawn on the diagram using Draw Item on Diagram function 164. At error checker 161, should an error condition be determined to exist, Enter Error Condition function 162 is invoked and function 135 exited at 165.

Referring again to Figure 4, function 140 gives the user the option of compiling the input into the repository only. Function 140 is detailed in Figure 7. After selecting function 140, Compiling into Repository Only, the system combines all text from the edit windows at 141 and parses the input data using Parse function 142. Error checker 143 determines if an error has been made in the textual input. If errors have been made, they are shown at error window 145. In the event no errors were incurred, successful compilation is shown in the status bar at 144. An iterative process is detailed at 146. Each of the lists generated, the object list, fact list and constraint list, is searched record by record. Each record is retrieved from it's respective list at 147 and its status tested at 148. In the event the record has been changed, function 149, which updates the record to the repository, is invoked. In the event the record has not been changed, the list pointer is incremented at 150 and a new record is retrieved from the list. In the event the record is new, function 149 is again invoked after which the record type is tested at 152. If the new record is an object, function 152, which allocates a new object, is invoked. In the event the new record is a fact, a new fact is allocated using function 154. If the new record type is a constraint, a new constraint is allocated at 155. Following any of these allocations, the list pointer is again incremented at 150.

Parse function 142 of Figures 6 and 7 is detailed at Figure 8. After Parse function 142 is invoked, it invokes Scan function 170 to retrieve a token. After the token is retrieved, it is partitioned into sections at 171. An iterative loop through each of the tokens is set up at 172. Each token is tested for its type at 173. In the event the section is an object, function 174, which looks for object specifications, is invoked. In the event the section is a fact, function 175, which looks for fact

specifications is invoked. In the event the section being tested is a constraint, function 176 is invoked which looks for constraint specifications. After either functions 174, 175 or 176 have been invoked, a test is made to determine if the section tested is the terminal section. If not, the next section is subjected to the same test. In the event the section is a terminal section, Parse function 142 is exited at 179.

Scan function 170 of Figure 8 is detailed at Figure 9. After Parse function 142 invokes Scan function 170, it reads characters from the input source text at 180. At 181, the function matches each character sequence with a token specification shown as regular expressions in the Lexical Analyzer Declarations at 181, after which the token is returned at 182.

Function 162, Enter Error Condition which was previously invoked at Figure 6, is detailed at Figure 10. For each error encountered, the system will retrieve the character position and line number where the error occurred. This function will then store the error number, error text and error context information in an error list.

Referring again to Figure 8, function 174, Look for object Specification, is detailed at Figure 11. When function 174 is invoked, the object is syntactically specified by any "Object Declaration" at 190. The function continues to "parse" and "scan". A determination is made at 191 as to whether it is still an object section. If not, the function exits at 192. If the object is still in the object section, it breaks the textual components into structures for storing in the object list at 193. At 194, the object status is tested. If it is a new object, function 220 which allocates a new object, is invoked, If the object already exists in an object list, function 174 is exited at 198. If the object exists in the repository but it is not in the object list, the system reads the structure at 196 from the repository and puts it in the object list. After either function 196 or 220 is invoked, the function is again exited at 198.

Function 175, Look for Fact Specification, previously invoked at Figure 8, is detailed in Figure 12. After function 175 is invoked, the fact is syntactically specified by a "fact declaration". The system continues to "parse" and "scan". A determination is made at 201 as to whether the fact is still in the fact section. If not, function 175 is exited at 202. If it is still in the fact section, the system breaks the textual components into structures for storing in a fact list at 203 after which the

fact status is tested at 204. If the fact is a new fact, function 230 is invoked, which allocates a new fact after which fact parsing is exited at 206. If the fact already exists in a fact list, the function 175 again exits at 206. If the fact exists in the repository but is not in the fact list, the system reads the structure at 205 from the repository and puts it in the fact list, after which function 175 exits once again at 206.

Function 176, also previously invoked at Figure 8, is detailed at Figure 13. When function 176 is invoked to look for constraint specifications, a constraint is syntactically specified by a "constraint declaration" at 210. The system continues to "parse" and "scan". At 211 a determination is made if the constraint is still in the constraint section. If not, function 176 is exited at 212. If the constraint is still in the constraint section, the system breaks the textural components into structures for storing in the constraint list at 213. After which the constraint status is tested at 214. If a new constraint, function 240 is invoked, which allocates a new constraint. If the constraint already exists in the constraint list, function 176 is exited at 217. If the constraint exists in the repository but it is not in the constraint list, the function reads the structure from the repository and puts it in the constraint list at 216. After either function 240 or process 216 is accomplished, function 176 exits at 217.

Function 220, previously invoked in Figure 11, is detailed in Figure 14. Function 220 creates an empty object structure at 221, enters the name and all object attributes into the structure at 222, places the object structure in the object list at 223, and exits at 224.

Function 230, previously invoked in Figure 12, is detailed in Figure 15. Function 230 creates an empty fact structure at 231. At 232 the function puts the predicate text, the objects involved and internal constraint information into the structure. At 233, the function places the fact structure into the fact list and exits at 234.

Function 240, previously invoked at Figure 13, is detailed in Figure 16. Function 240 allocates a new constraint as follows: it creates an empty constraint structure at 241. At 242 it puts the constraint information (role positions, predicate IDs) into the structure. At 243 the function places the constraint structure into the constraint list and exits at 244.

Function 149, previously invoked in both Figures 7 and 8, is detailed in Figure 17. After function 149 is invoked, an updated record exists in a fact list, an object list or a constraint list, as shown at 250. At 251, the record type is tested. If the record is an object, the object structure is sent the object update function at 252. If the record is a fact, the fact structure is sent to the fact update function at 254 and if a constraint structure is sent to a constraint update function at 253. After any of the aforementioned update functions is accomplished, function 149 is exited at 256.

Fact Tree Formation

Referring to Figure 18, the second option possible from function selection 51 is initiation of Fact Tree Specification 300, which is detailed at Figure 19.

Referring to Figure 19, the fact-tree is formed at 300. An example of a fact tree is:

Person (= Mr Smith) .....restriction
|-- that person lives at an address
|-- that has a phone number.....noun
|-- that studies a subject
|-- that is taught by a Person
|.....phrase"

Each node in the fact tree has a noun (e.g. Person).

Each node in the fact tree may have a restriction (e.g. = Mr Smith).

Each non-root node in the fact tree (all but the very top node) has a phrase (e.g. is taught by).

The root of the tree is then assigned to the variable *Root* at 301. In this case, the shaded node (Person) is assigned to *Root*.

If the fact tree is to be mapped to an SQL query, *Root* is passed as the parameter to the function Fact_Tree_To_SQL 400. The return value of this function will be an SQL query. Fact_Tree_To_SQL is described using functions 400 to 465.

If the fact tree is to be mapped to an English description, *Root* is passed as the parameter to the function Fact_Tree_To_Description (500). This function has no return value. The result of Fact Tree To Description will be to print out the description of the tree. Fact_Tree_To_Description is described using functions 500 to 535.

Tree Interpreter

The present invention also provides a Tree Interpreter, invoked as "Fact_Tree_to_Description", a detailed description of which follows. Referring to Figure 25, Fact_Tree_To_Description (500) is a recursive function that takes a node of a fact tree as input and returns a description of the query represented by that tree or sub-tree. The parameter *Root* is the node on which the function is to operate.

Function 501 assigns some working variables. *Root* is the root of a tree of subtree that may or may not contain a parent and may or may not contain children. For example, if the shaded node in the example tree (Parent) was passed to Fact_Tree_To_Description, there would be no parent, and three children. *Parent* is assigned *Root's* parent (in this case NULL). *Nodes* is assigned the number of *Root's* children (three). *Child [i .. Nodes]* is an array which is assigned *Root's* children (the three children of Person -- "that lives at Address", "that has a phone number", and "that studies a subject".

Next, the temporary variable *Text* is assigned the "description" of the *Root.* (502-504). If *Root* has no parent (it is the root of the fact tree), the *Root* is passed to the function 503 Create_Text_For_Root (503), otherwise *Root* is passed to Function 507, Create_Text_For_Node. The return value of both of these functions is the description of *Root.*

At 505, *Text* is then printed out followed by a carriage return. If *Root* referred to the Person node, Function 503, Create Text For Root would have been used and *Text* would be "For all Person(s) (where Person = Mr Smith) show:" If *Root* referred to the "that studies Subject" node, function 504, Create Text For Node would have been used and *Text* would be "the Subject(s) that the Person studies, and for those Subject(s) show: ".

The next step is to recursively process *Root's* children using a depth-first-search algorithm, detailed in functions 505-510. *Nodes* is the number of *Root's* children and *Child [i .. Nodes]* are *Root's* actual children.

The variable *I* is used as a counter variable. It is initially assigned to 1 at 506. If *I* is greater than *Nodes* (there are no more children to process), this instantiation of Fact_Tree_To_Description is complete (507-508) otherwise,

Fact_Tree_To_Description is invoked for *Child[I]* at 509. *I* is incremented at 510 and the loop continues (507) until there are no more children to process.

The result of Fact_Tree_To_Description applied to the Person node in the example tree would be:

Output	Processing
For all Person(s) (where Mr. Smith) show:	Fact_Tree To .Description called on Person
the Address that the Person lives at	Fact_Tree To .Description called on that lives at Address
the Phone Number that the Person has	Fact Tree To .Description called on that has phone number
the Subjects that the Person studies, and for those subjects show:	Fact_Tree To .Description called on that studies subject
The Person(s) that the Subject is taught by	Fact Tree To Description called on that Is taught by Person

Function 503, is invoked in Figure 25 and Create_Text_For_Root, takes the root node of a fact tree as an argument *(Root)* and returns the description of that node.

At 520, Root contains a noun (e.g. Person); this is assigned to *Noun. Root* may also contain a *Restriction* (e.g. = Mr Smith); this is assigned to *Restriction. The* variable *Text* is assigned 'For all' + noun + '(s)' (e.g. 'For all Person(s)').

If Root contains a *Restriction*, that restriction is added to *Text* at 521 and 522. (e.g. *Text* = "For all Person(s)(where Person = Mr Smith)")

"show": 'is added to *Text* at 523 and

Text is returned at 524 (e.g. *Text* = "For all Person(s)(where Person = Mr Smith) show:").

Referring back to Figure 25, Create_Text_For_Node (504), which takes a non-root node of a fact tree as an argument (*Root*) and returns the description of that node is detailed at Figure 27. For example, the node 'that studies a Subject' could be passed as an argument.

At 530 *Root* contains a noun (e.g. Subject), this is assigned to *Noun. Root* contains a phrase (e.g. 'studies') this is assigned to *Phrase. Root's* parent contains a noun (e.g. Person), this is assigned to *Parent-Noun.* Root may also contain a

Restriction (e.g. = Mr Smith) this is assigned to *Restriction. The* variable *Text* is assigned 'the' + *noun* + *'(s)* that the ' + *Parent-Noun* + ' ' + *Phrase* (e.g. 'the Subject(s) that the Person studies').

If *Root* contains a *Restriction,* that restriction is added to *Text* at 531 and 532.

If *Root* has children', and for those' + *Noun* + ':(s) show:' is added to *Text* at 533 and 534.

At 535, *Text is* returned (e.g. *Text* = 'the Subject(s) that the Person studies, and for those Subject(s) show:').

Query Mapper

The present invention also provides a Query Mapper, invoked as "Fact_Tree_to_SQL_Query", a detailed description of which follows. Referring back to Figure 19, function 400 is detailed at Figure 20. This function takes the root node of a fact tree as input and returns an equivalent SQL query. The parameter *Root* is the root of the tree on which the function is to operate.

Each non-root node in the fact tree (all but the very top node) has a relational mapping associated with it. The relational mapping specifies the node's representation in a relational database. By way of example, for an example set of facts:

Person lives at address
Person has Phone Number
Person studies Subject
Subject is taught by Person

if the relational database associated with an example fact tree is:

Person _Table:	(Person, Address)
Phone _Table:	(Person, Phone Number)
Studies _Table:	(Person, Subject Studied)
Subject_Table:	(Subject, Teacher Person)

The associated RMAP mappings would be:

FACT	TABLE	FIRST NOUN COLUMN	SECOND NOUN COLUMN
Person lives at address	Person_Table	Person	Address
Person has Phone Number	Phone_table	Person	Phone Number
Person studies Subject	Studies_Table	Person	Subject Studies
Subject is taught by Person	Subject_Table	Person	Teacher Person

Table denotes the table in which the fact is stored. First Noun Column denotes the column for the node's parent. Second Noun Column denotes the column for the node's noun.

For example, the mapping associated with *(Person) lives at Address* means that all addresses are stored in the Person Table table, with the people who live at them in the Person column and the actual addresses in the Address column.

The predicate mappings are derived by an algorithm similar to the one described in McCormack & Halpin, *Automated Mapping of Conceptual Schemas to Relational Schemas*, Proc CAiSE 93, Sorbonne University, Paris, 1993.

An SQL query contains three parts, a *SelectList*, *a From List*, and a *WhereClause*. These are gradually build up using recursive calls to Node_To_SQL.

Initially, *SelectList*, *FromList*, and *WhereClause* are set to the empty string (" ") at 401.

At 402, Node _To_SQL is called, with *Root* as its parameter, to build up the query.

At 403, 404 and 405, *SelectList*, *FromList*, and *WhereClause* are respectively formatted. At 406, the query is assembled as the result of the function and returned at 407.

Node_To_SQL is invoked at 402. This is a recursive function that maps a node of a fact tree into an SQL query. Successive calls to this function build up the *SelectList*, *FromList*, and *WhereClause* variables.

Node_To_SQL has three parameters. The first, *Root*, is the root of the tree or sub tree being mapped. The second and third parameters are the table and column used to join the query for *Root's* sub tree to the rest of the query.

At 410, *Parent* is the parent of *Root*, *Nodes* is the number of children of *Root* and *Child [i .. Nodes]* are the children of *Root*.

If *Root* has no children, no processing is required so the function simply returns (411, 412).

Otherwise, the children of *Root* are added to the query as follows:

If *Root* has a parent, it needs to be joined into the query using the *JoinTable* and *JoinColumn* function 413 and 414; and *Root* needs to be added to the select list of the query at 415.

To join all of *Root's* children into the query, they are processed sequentially as follows: The counter variable i is initialized to 1 at 416. The value of nodes is checked at 417, and if i is less than nodes Add_Selector_2 (node) is invoked at 419. Each child is added to the select-list using Add_Selector_2 at 419, and that child's children are added into the query using recursive calls to function 421, Node_To_SQL. The selector for each node is used to join the subtree's queries together at 420. Create_Join, invoked at 414 is detailed at Figure 22. This function joins a subquery to the main query by adding an inner join to the where-clause. The join is based in the first noun in the passes node *(Node)* and the passed parameters.

Referring back to Figure 21, function 415, Add_Selector_1 is detailed at Figure 23. This function adds *Node's* table to the *FromList* and the column for *Node's* first noun to the SelectList.

Referring again to Figure 21, function 419, Add_Selector_2 is detailed at Figure 24. This function adds *Node's* table to the *FromList* and the column for *Node's* second noun to the *SelectList.*

It is to be understood that the above-described embodiments are merely illustrative of some of the many specific embodiments which represent applications of the principles of the present invention. Clearly, numerous variations can be readily devised by those skilled in the art without departing from the scope of the present invention.

CLAIMS

What is claimed is:

1. Apparatus including a general purpose programmable digital computer, said computer having central processing means, bus means, display means, data entry means, memory means, data storage means and graphical user interface, for specifying database designs, said apparatus further comprising;

diagram means for producing a diagram on said display means;

cursor control means for controlling movement of a cursor over said diagram;

repository means further comprising relational database means implemented on said computer;

text input means for entering into an edit window, text using said data entry means, items of said text including data objects, facts about said data objects and constraints on said data objects;

user-selectable text validation means for validating said text;

user-selectable translation means for translating said text from said text input means into said diagram; and

user-selectable first compilation means for compiling said text only into said repository means.

2. The apparatus of Claim 1 wherein said text validation means further comprises:

first text combining means for combining all text from said edit window;

first parsing means, responsive to said first text combining means, for parsing said combined text into objects, facts and constraints;

first list means, responsive to said first parsing means, for storing said objects, facts and constraints in said memory means;

first error checking means, responsive to said first parsing means, for determining if an error exists in said objects, facts or constraints following parsing by said first parsing means; and

first indicator means responsive to said first error checking means for indicating whether or not an error was determined to exist by said first error checking means.

3. The apparatus of Claim 1 wherein said translation means further comprises:

capture means responsive to a first selector means of said cursor control means for capturing said text item from within said edit window;

item test means, responsive to said capture means, for testing whether said text item is an object, a fact or a constraint;

first cursor change means, responsive to said item test means for changing said cursor to reflect whether said text item is an object, a fact or a constraint;

cursor release means, responsive to said first selector means of said cursor control means, for releasing said cursor over said diagram;

text collection means for collecting said text at said edit window;

second parsing means responsive to said text collection means for parsing said text item into objects, facts and constraints;

second list means, responsive to said second parsing means for storing said objects, facts and constraints into said memory means;

first update means for updating repository by copying said objects, facts and constraints from said second list means and storing said objects, facts and constraints into said repository means as records; and

said diagram means further responsive to said cursor control means, said capture means, said item test means, said first cursor change means, said cursor release means, and said second parsing means for drawing said objects, facts and constraints on said diagram.

4. The apparatus of Claim 3 further comprising:

cursor rotation means, responsive to a second selector means of said cursor control means, for rotating said cursor 90° each time said second selector means of said cursor control means is actuated; and

said diagram means, further responsive to said cursor rotation means for drawing said item on said diagram.

5. The apparatus of Claim 3, further comprising edit window clearing means, responsive to movement of said cursor over said diagram, for causing said edit window to disappear from said display means while said cursor is being moved.

6. The apparatus of Claim 3, further comprising;

second error checking means, responsive to position of said cursor, for determining if said cursor is over said diagram;

second indicator means, responsive to said second error checking means, for indicating when said cursor is not over said diagram;

third error checking means for checking if one or more errors exists in any of said objects, facts or constraints after operation of said second parsing means; and

said diagram means, further responsive to said second error checking means for drawing said item on said diagram.

7. The apparatus according to Claim 2 wherein said first parsing means further comprises:

first scan means for returning a token corresponding to said text item;

first partition means, responsive to said first scan means, for partitioning into sections each token retrieved by said scan means;

first token test means, responsive to said first partition means, for testing whether each of said sections is an object, a fact or a constraint; and

first terminal test means, responsive to said first token test means, for testing if each of said sections is a terminal section.

8. The apparatus of Claim 7 further comprising, upon a determination by said first token test means that said section is an object:

first iterative parse means for iteratively continuing to parse said section;

first iterative scan means for iteratively continuing to scan said section;

first structuring means, responsive to a determination by said first iterative parse means and said first iterative scan means that said section is still an object, for structuring said object for storage in said first list means;

first object status test means, responsive to said first structuring means for testing whether said object was previously stored in said first list means; whether said object was previously stored in said repository means but not stored in said first list means or whether said object is newly created;

first object listing means, responsive to a determination by said object first status test means that said object was previously stored in said repository means but not stored in said first list means, for copying said object from said repository means and storing said object in said first list means; and

first new object allocation means, responsive to a determination by said first object status test means that said object is newly created, for allocating said newly created object to said repository means.

9. The apparatus of Claim 7 further comprising, upon a determination by said first token test means that said section is a fact:

second iterative parse means for iteratively continuing to parse said section;

second iterative scan means for iteratively continuing to scan said section;

second structuring means, responsive to a determination by said second iterative parse means and said second iterative scan means that said section is still a fact, for structuring said fact for storage in said first list means;

first fact status test means, responsive to said second structuring means for testing whether said fact was previously stored in said first list means; whether said fact was previously stored in said repository means but not stored in said first list means or whether said fact is newly created;

first fact listing means, responsive to a determination by said first fact status test means that said object was previously stored in said repository means but not stored in said first list means, for copying said fact from said repository means and storing said fact in said first list means; and

first new fact allocation means, responsive to a determination by said first fact status test means that said fact is newly created, for allocating said newly created fact to said repository means.

10. The apparatus of Claim 7 further comprising, upon a determination by said first token test means that said section is a constraint:

third iterative parse means for iteratively continuing to parse said section;

third iterative scan means for iteratively continuing to scan said section;

third structuring means, responsive to a determination by said third iterative parse means and said third iterative scan means that said section is still a constraint, for structuring said constraint for storage in said first list means;

first constraint status test means, responsive to said third structuring means for testing whether said constraint was previously stored in said first list means; whether said constraint was previously stored in said repository means but not stored in said first list means or whether said constraint is newly created;

first constraint listing means, responsive to a determination by said first constraint status test means that said object was previously stored in said repository means but not stored in said first list means, for copying said constraint from said repository means and storing said constraint in said first list means; and

first new constraint allocation means, responsive to a determination by said first constraint status test means that said constraint is newly created, for allocating said newly created constraint to said repository means.

11. The apparatus according to Claim 3 wherein said second parsing means further comprises:

second scan means for returning a token corresponding to said text item;

second partition means, responsive to said second scan means, for partitioning into sections each token retrieved by said second scan means;

second token test means, responsive to said second partition means, for testing whether each of said sections is an object, a fact or a constraint; and

second terminal test means, responsive to said second token test means, for testing if each of said sections is a terminal section.

12. The apparatus of Claim 11 further comprising, upon a determination by said second token test means that said section is an object:

fourth iterative parse means for iteratively continuing to parse said section;

fourth iterative scan means for iteratively continuing to scan said section;

fourth structuring means, responsive to a determination by said fourth iterative parse means and said fourth iterative scan means that said section is still an object, for structuring said object for storage in said second list means;

second object status test means, responsive to said fourth structuring means for testing whether said object was previously stored in said second list means; whether said object was previously stored in said repository means but not stored in said second list means or whether said object is newly created;

second object listing means, responsive to a determination by said object status test means that said object was previously stored in said repository means but not stored in said second list means, for copying said object from said repository means and storing said object in said second list means; and

second new object allocation means, responsive to a determination by said object status test means that said object is newly created, for allocating said newly created object to said repository means.

13. The apparatus of Claim 11 further comprising, upon a determination by said second token test means that said section is a fact:

fifth iterative parse means for iteratively continuing to parse said section;

fifth iterative scan means for iteratively continuing to scan said section; and

fifth structuring means, responsive to a determination by said fifth iterative parse means and said fifth iterative scan means that said section is still a fact, for structuring said fact for storage in said second list means;

second fact status test means, responsive to said fifth structuring means for testing whether said fact was previously stored in said second list means; whether said fact was previously stored in said repository means but not stored in said second list means or whether said fact is newly created;

second fact listing means, responsive to a determination by said second fact status test means that said object was previously stored in said repository means but not stored in said second list means, for copying said fact from said repository means and storing said fact in said second list means; and

second new fact allocation means, responsive to a determination by said second fact status test means that said fact is newly created, for allocating said newly created fact to said repository means.

14. The apparatus of Claim 11 further comprising, upon a determination by said second token test means that said section is a constraint:

sixth iterative parse means for iteratively continuing to parse said section;

sixth iterative scan means for iteratively continuing to scan said section;

sixth structuring means, responsive to a determination by said sixth iterative parse means and said sixth iterative scan means that said section is still a constraint, for structuring said constraint for storage in said second list means;

second constraint status test means, responsive to said sixth structuring means for testing whether said constraint was previously stored in said second list means; whether said constraint was previously stored in said repository means but not stored in said second list means or whether said constraint is newly created;

second constraint listing means, responsive to a determination by said second constraint status test means that said object was previously stored in said repository means but not stored in said second list means, for copying said constraint from said repository means and storing said constraint in said second list means; and

second new constraint allocation means, responsive to a determination by said second constraint status test means that said constraint is newly created, for allocating said newly created constraint to said repository means.

15. The apparatus of Claim 1 wherein said first compilation means further comprises:

second means for combining all text from said edit window into an item;

third parsing means for parsing said item into objects, facts and constraints;

third list means, responsive to said third parsing means for storing said objects, facts and constraints into said memory means as a record;

record status test means for testing said record as being a newly created record, a changed record or an unchanged record;

second update means, responsive to a determination by said record status test means that said record is changed, for updating said repository means by copying said objects, facts and constraints from said third list means and storing said objects, facts and constraints into said repository means as records;

third update means, responsive to a determination by said record status test means that said record is newly created, for updating said repository means by copying said objects, facts and constraints from said third list means and storing said objects, facts and constraints into said repository means as records;

record type test means;

third new object allocation means, responsive to a determination by said record type test means that said record is a object and to a determination by said record status test means that said record is newly created, for allocating said object to said repository means;

third new fact allocation means, responsive to a determination by said record type test means that said record is a fact and to a determination by said record status test means that said record is newly created, for allocating said fact to said repository means; and

third new constraint allocation means, responsive to a determination by said record type test means that said record is a constraint and to a determination by said record status test means that said record is newly created, for allocating said constraint to said repository means.

16. The apparatus according to Claim 15 wherein said third parsing means further comprises:

third scan means for returning a token corresponding to said text item;

third partition means, responsive to said third scan means, for partitioning into sections each token retrieved by said third scan means;

third token test means, responsive to said third partition means, for testing whether each of said sections is an object, a fact or a constraint; and

third terminal test means, responsive to said third token test means, for testing if each of said sections is a terminal section.

17. The apparatus of Claim 16 further comprising, upon a determination by said third token test means that said section is an object:

seventh iterative parse means for iteratively continuing to parse said section;

seventh iterative scan means for iteratively continuing to scan said section;

seventh structuring means, responsive to a determination by said seventh iterative parse means and said seventh iterative scan means that said section is still an object, for structuring said object for storage in said third list means;

third object status test means, responsive to said seventh structuring means for testing whether said object was previously stored in said third list means; whether said object was previously stored in said repository means but not stored in said third list means or whether said object is newly created;

third object listing means, responsive to a determination by said third object status test means that said object was previously stored in said repository means but not stored in said third list means, for copying said object from said repository means and storing said object in said third list means; and

third new object allocation means, responsive to a determination by said third object status test means that said object is newly created, for allocating said newly created object to said repository means.

18. The apparatus of Claim 16 further comprising, upon a determination by said third token test means that said section is a fact:

eighth iterative parse means for iteratively continuing to parse said section;

eighth iterative scan means for iteratively continuing to scan said section; and

eighth structuring means, responsive to a determination by said eighth iterative parse means and said eighth iterative scan means that said section is still a fact, for structuring said fact for storage in said third list means;

third fact status test means, responsive to said eighth structuring means for testing whether said fact was previously stored in said third list means; whether said fact was previously stored in said repository means but not stored in said third list means or whether said fact is newly created;

third fact listing means, responsive to a determination by said third fact status test means that said object was previously stored in said repository means but not stored in said third list means, for copying said fact from said repository means and storing said fact in said third list means; and

third new fact allocation means, responsive to a determination by said third fact status test means that said fact is newly created, for allocating said newly created fact to said repository means.

19. The apparatus of Claim 18 further comprising, upon a determination by said third token test means that said section is a constraint:

ninth iterative parse means for iteratively continuing to parse said section;

ninth iterative scan means for iteratively continuing to scan said section;

ninth structuring means, responsive to a determination by said ninth iterative parse means and said ninth iterative scan means that said section is still a constraint, for structuring said constraint for storage in said third list means;

third constraint status test means, responsive to said ninth structuring means for testing whether said constraint was previously stored in said third list means; whether said constraint was previously stored in said repository means but not stored in said third list means or whether said constraint is newly created;

third constraint listing means, responsive to a determination by said constraint status test means that said object was previously stored in said repository

means but not stored in said third list means, for copying said constraint from said repository means and storing said constraint in said third list means; and

third new constraint allocation means, responsive to a determination by said constraint status test means that said constraint is newly created, for allocating said newly created constraint to said repository means.

20. The apparatus of Claim 4, wherein said cursor rotation means rotates said cursor 90° in a clockwise direction each time said second selector means of said cursor control means is actuated.

21. The apparatus of Claim 4, wherein said cursor rotation means rotates said cursor 90° in a counterclockwise direction each time said second selector means of said cursor control means is actuated.

22. Method for specifying database designs on a general purpose programmable digital computer, said computer having central processing means, bus means, display means, data entry means, memory means, data storage means, graphical user interface and repository means further comprising relational database means implemented on said computer; said method comprising the steps of;

producing a diagram on said display means;

moving a cursor over said diagram;

entering, into an edit window, text from said data entry means, items of said text including data objects, facts about said data objects and constraints on said data objects;

validating said text responsive to a user-selectable option;

translating said text from said text input means into said diagram responsive to a user-selectable option; and

compiling said text only into said repository means responsive to a user-selectable option.

23. The method of Claim 22 wherein said text validation method further comprises the steps of:

combining all text from said edit window using a first text combining method;

parsing said combined text into objects, facts and constraints using first parsing method, responsive to said first text combining method;

storing said objects, facts and constraints in said memory means; using first list means, responsive to said first parsing method;

determining if an error exists in said objects, facts or constraints following parsing by said first parsing method; using first error checking method, responsive to said first parsing method, and

indicating whether or not an error was determined to exist by said first error checking method using first indicator method responsive to said first error checking method.

24. The method of Claim 22 wherein said translation method further comprises the steps of:

capturing said text item from within said edit window using capture method responsive to a first selector method of said cursor control means;

testing whether said text item is an object, a fact or a constraint using item test method, responsive to said capture method;

changing said cursor to reflect whether said text item is an object, a fact or a constraint using first cursor change means, responsive to said item test method for;

releasing said cursor over said diagram;

collecting said text at said edit window;

parsing said text item into objects, facts and constraints;

storing said objects, facts and constraints into said memory means;

updating said repository by copying said objects, facts and constraints from said second list means and storing said objects, facts and constraints into said repository means as records; and

drawing said objects, facts and constraints on said diagram using said diagram means further responsive to said cursor control means, said capture method, said item test method, said first cursor change method, said cursor release method, and said second parsing method.

25. The method of Claim 24, further comprising the steps of:

rotating said cursor 90° each time said second selector method of said cursor control means is actuated; and

drawing said item on said diagram, said diagram method further responsive to said cursor rotation method.

26. The method of Claim 24, further comprising the step of causing said diagram to disappear from said display means while said cursor is being moved.

27. The method of Claim 24, further comprising;

determining if said cursor is over said diagram using second error checking method, responsive to position of said cursor;

second indicator method, responsive to said second error checking method, for indicating when said cursor is not over said diagram;

third error checking method for checking if one or more errors exists in any of said objects, facts or constraints after operation of said second parsing method; and

drawing said item on said diagram using said diagram method, further responsive to said second error checking method.

28. The method according to Claim 23 wherein said first parsing method further comprises:

returning a token corresponding to said text item using first scan method;

partitioning into sections each token retrieved by said scan method first partition method, responsive to said first scan method;

testing whether each of said sections is an object, a fact or a constraint using first token test method, responsive to said first partition method; and

testing if each of said sections is a terminal section using a first terminal test method, responsive to said first token test method.

29. The method of Claim 28, upon a determination by said first token test method that said section is an object, further comprising the steps of:

iteratively continuing to parse said section using a first iterative parse method;

iteratively continuing to scan said section using a first iterative scan method;

structuring said object for storage in said first list means using a first structuring method, responsive to a determination by said first iterative parse method and said first iterative scan method that said section is still an object.

testing, with a first object status test method, responsive to said first structuring method for whether said object was previously stored in said first list means; whether said object was previously stored in said repository means but not stored in said first list means or whether said object is newly created;

copying said object from said repository means using a first object listing means, responsive to a determination by said object first status test method that said object was previously stored in said repository means but not stored in said first list means and storing said object in said first list means; and

allocating said new object to said repository means using a first newly created object allocation method, responsive to a determination by said first object status test method that said object is newly created.

30. The method of Claim 28, upon a determination by said first token test method that said section is a fact, further comprising the steps of:

iteratively continuing to parse said section using a second iterative parse method;

iteratively continuing to scan said section using a second iterative scan method;

structuring said fact for storage in said first list means using a second structuring method, responsive to a determination by said second iterative parse method and said second iterative scan method that said section is still a fact;

testing whether said fact was previously stored in said first list means; whether said fact was previously stored in said repository means but not stored in said first list means or whether said fact is newly created using a first fact status test method, responsive to said second structuring method;

copying said fact from said repository means and storing said fact in said first list means; using a first fact listing means, responsive to a determination by said first fact status test method that said object was previously stored in said repository means but not stored in said first list means; and

allocating said newly created fact to said repository means using a first newly created fact allocation method, responsive to a determination by said first fact status test method that said fact is newly created.

31. The method of Claim 28, upon a determination by said first token test method that said section is a constraint, further comprising the steps of:

iteratively continuing to parse said section using a third iterative parse method;

iteratively continuing to scan said section using a third iterative scan method;

structuring said constraint for storage in said first list means using a third structuring method, responsive to a determination by said third iterative parse method and said third iterative scan method that said section is still a constraint;

testing whether said constraint was previously stored in said first list means; whether said constraint was previously stored in said repository means but not stored in said first list means or whether said constraint is newly created using a first constraint status test method, responsive to said third structuring method;

copying said constraint from said repository means and storing said constraint in said first list means using a first constraint listing method, responsive to a determination by said first constraint status test method that said object was previously stored in said repository means but not stored in said first list means; and

allocating said newly created constraint to said repository means using a first new constraint allocation method, responsive to a determination by said first constraint status test method that said constraint is newly created.

32. The method according to Claim 24 wherein said second parsing method further comprises the steps of:

returning a token corresponding to said text item using a second scan method;

partitioning into sections each token retrieved by said second scan method using a second partition method, responsive to said second scan method;

testing whether each of said sections is an object, a fact or a constraint using a second token test method, responsive to said second partition method; and

testing if each of said sections is a terminal section using a second terminal test method, responsive to said second token test method.

33. The method of Claim 32, upon a determination by said second token test method that said section is an object, further comprising the steps of:

iteratively continuing to parse said section using a fourth iterative parse method;

iteratively continuing to scan said section using a fourth iterative scan method;

structuring said object for storage in said second list method using a fourth structuring method, said fourth structuring method responsive to a determination by said fourth iterative parse method and said fourth iterative scan method that said section is still an object.

testing, using a second object status test method, responsive to said fourth structuring method, whether said object was previously stored in said second list means; whether said object was previously stored in said repository means but not stored in said second list means or whether said object is newly created;

copying said object from said repository means and storing said object in said second list means using a second object listing method, responsive to a determination by said object status test method that said object was previously stored in said repository means but not stored in said second list means; and

allocating said newly created object to said repository means using a second new object allocation method, responsive to a determination by said object status test method that said object is newly created.

34. The method of Claim 32 further comprising, upon a determination by said second token test method that said section is a fact, the following steps:

iteratively continuing to parse said section using a fifth iterative parse method;

iteratively continuing to scan said section using a fifth iterative scan method; and

structuring said fact for storage in said second list means using a fifth structuring method, responsive to a determination by said fifth iterative parse method and said fifth iterative scan method that said section is still a fact;

testing whether said fact was previously stored in said second list means; whether said fact was previously stored in said repository means but not stored in said second list means or whether said fact is newly created using a second fact status test method, responsive to said fifth structuring method;

copying said fact from said repository means and storing said fact in said second list means using a second fact listing method, responsive to a determination by said second fact status test method that said object was previously stored in said repository means but not stored in said second list means; and

allocating said newly created fact to said repository means using a second new fact allocation method, responsive to a determination by said second fact status test method that said fact is newly created.

35. The method of Claim 32, upon a determination by said second token test method that said section is a constraint, further comprising the steps of:

iteratively continuing to parse said section using a sixth iterative parse method;

iteratively continuing to scan said section using a sixth iterative scan method;

structuring said constraint for storage in said second list means using a sixth structuring method, responsive to a determination by said sixth iterative parse method and said sixth iterative scan method that said section is still a constraint;

testing whether said constraint was previously stored in said second list means; whether said constraint was previously stored in said repository means but not stored in said second list means or whether said constraint is newly created using a second constraint status test method, responsive to said sixth structuring method;

copying said constraint from said repository means and storing said constraint in said second list means using a second constraint listing means, responsive to a determination by said second constraint status test method that said object was previously stored in said repository means but not stored in said second list means; and

allocating said newly created constraint to said repository means using a second new constraint allocation method, responsive to a determination by said second constraint status test method that said constraint is newly created.

36. The method of Claim 22 wherein said first compilation method further comprises the steps of:

combining all text from said edit window into an item using a second text combining method;

parsing said item into objects, facts and constraints using a third parsing method;

storing said objects, facts and constraints into said memory means as a record using a third list means, responsive to said third parsing method;

testing said record as being a newly created record, a changed record or an unchanged record using a record status test method;

updating said repository means by copying said objects, facts and constraints from said third list means and storing said objects, facts and constraints into said repository means as records using a second update method, responsive to a determination by said record status test method that said record is changed;

updating said repository means by copying said objects, facts and constraints from said third list means and storing said objects, facts and constraints into said repository means as records using a third update method, responsive to a determination by said record status test method that said record is newly created;

testing the record type using a record type test method;

allocating said object to said repository means using a third new object allocation method, responsive to a determination by said record type test method that said record is an object and to a determination by said record status test method that said record is newly created;

allocating said fact to said repository means using a third new fact allocation method, responsive to a determination by said record type test method that said record is a fact and to a determination by said record status test method that said record is newly created; and

allocating said constraint to said repository means, using a third new constraint allocation means, responsive to a determination by said record type test

means that said record is a constraint and to a determination by said record status test means that said record is newly created.

37. The method according to Claim 36 wherein said third parsing method further comprises the steps of:

returning a token corresponding to said text item using a third scan method;

partitioning into sections each token retrieved by said third scan method using a third partition method, responsive to said third scan method;

testing whether each of said sections is an object, a fact or a constraint using a third token test method, responsive to said third partition method; and

testing if each of said sections is a terminal section using a third terminal test method, responsive to said third token test method.

38. The method of Claim 37, upon a determination by said third token test method that said section is an object, further comprising the steps of:

iteratively continuing to parse said section using a seventh iterative parse method;

iteratively continuing to scan said section using a seventh iterative scan method;

structuring said object for storage in said third list means using a seventh structuring method, responsive to a determination by said seventh iterative parse method and said seventh iterative scan method that said section is still an object;

testing whether said object was previously stored in said third list means; whether said object was previously stored in said repository means but not stored in said third list means or whether said object is newly created using a third object status test method, responsive to said seventh structuring method;

copying said object from said repository means and storing said object in said third list means using a third object listing method, responsive to a determination by said third object status test method that said object was previously stored in said repository means but not stored in said third list means; and

allocating said newly created object to said repository means using a third new object allocation method, responsive to a determination by said third object status test method that said object is newly created.

39. The method of Claim 37, upon a determination by said third token test method that said section is a fact, further comprising the steps of:

iteratively continuing to parse said section using an eighth iterative parse method;

iteratively continuing to scan said section using an eighth iterative scan method;

structuring said fact for storage in said third list means using an eighth structuring method, responsive to a determination by said eighth iterative parse method and said eighth iterative scan method that said section is still a fact;

testing whether said fact was previously stored in said third list means; whether said fact was previously stored in said repository means but not stored in said third list means or whether said fact is newly created using a third fact status test method, responsive to said eighth structuring method;

copying said fact from said repository means and storing said fact in said third list means using a third fact listing method, responsive to a determination by said third fact status test method that said object was previously stored in said repository means but not stored in said third list means; and

allocating said newly created fact to said repository means using a third new fact allocation method, responsive to a determination by said third fact status test method that said fact is newly created.

40. The method of Claim 37, upon a determination by said third token test method that said section is a constraint further comprising the steps of:

iteratively continuing to parse said section using a ninth iterative parse method;

iteratively continuing to scan said section using a ninth iterative scan method;

structuring said constraint for storage in said third list means using a ninth structuring method, responsive to a determination by said ninth iterative

parse method and said ninth iterative scan method that said section is still a constraint;

testing whether said constraint was previously stored in said third list means; whether said constraint was previously stored in said repository means but not stored in said third list means or whether said constraint is newly created using a third constraint status test method, responsive to said ninth structuring method;

copying said constraint from said repository means and storing said constraint in said third list means using a third constraint listing method, responsive to a determination by said constraint status test method that said object was previously stored in said repository means but not stored in said third list means; and

allocating said newly created constraint to said repository means using a third new constraint allocation method, responsive to a determination by said constraint status test method that said constraint is newly created.

41. The method of Claim 25, wherein said cursor rotation method rotates said cursor 90° in a clockwise direction each time said second selector method of said cursor control means is actuated.

42. The method of Claim 25, wherein said cursor rotation method rotates said cursor 90° in a counterclockwise direction each time said second selector method of said cursor control means is actuated.

43. Apparatus including a general purpose programmable digital computer, said computer having central processing means, bus means, display means, data entry means, memory means, data storage means, and graphical user interface for describing in a natural language a query to a database, said apparatus further comprising;

diagram means for producing a diagram on said display means;

cursor control means for moving a cursor over said diagram;

repository means further comprising relational database means implemented on said computer;

fact tree formation means for forming a fact tree based on said query; and

fact tree description means for describing said fact tree in said natural language.

44. The apparatus of Claim 43, wherein said fact tree description means further comprises:

first variable assignment means for assigning variables based on said fact tree, said variables comprising root, parent, child and node, wherein said root is the root of said fact tree, said parent is the parent of said root, said child is the child of the root and said node is the number of the child;

parent test means for testing if a value of said parent is null;

root text creation means, responsive to a determination by said parent test means that said value of said parent is null, for creating text for said root;

node text creation means, responsive to a determination by said parent test means that a value of said parent is not null, for creating text for said node;

print means for printing, on said display means, said text created by said root text creation means and said node text creation means;

counter means for counting an iteration as an iteration value;

node test means for determining if said iteration value is equal to a value of said node; and

recursive means for recursively invoking said fact tree description means using depth-first search means.

45. The apparatus of Claim 44, wherein said root text creation means further comprises:

second variable assignment means for assigning variables based on said fact tree, said variables comprising first noun, first restriction and first text wherein said first noun is the noun in said root, said first restriction is the restriction in said root and said first text is equal to the value of said noun and a phrase indicating totality;

first restriction test means for testing if a value of said first restriction is null;

first text modification means, responsive to a determination by said first restriction test means that a value of said first restriction is not null, for

modifying said first text to be equal to said text and a word or phrase indicating presentation;

second text modification means, responsive to a determination by said first restriction test means that a value of said restriction is null, for modifying said text to be equal to said text and said noun and said restriction and a word or phrase indicating presentation; and

first text return means for returning said text as modified by said first or said second text modification means to said fact tree description means.

46. The apparatus of Claim 44, wherein said node text creation means further comprises:

third variable assignment means for assigning variables based on said fact tree, said variables comprising second noun, parent-noun, phrase, second restriction and second text wherein said second noun is a noun in said root, said parent-noun is a noun in said root's parent, said phrase is a phrase of said root, said restriction is a restriction in said root and said second text is equal to said second noun added to said parent-noun added to said phrase;

second restriction test means for testing if a value of said second restriction is null;

third text modification means, responsive to a determination by said second restriction test means that a value of said restriction is not null, for modifying said text to be equal to said second text added to said second noun added to said second restriction;

child testing means for testing if said root has said child;

fourth text modification means, responsive to a determination by said child test means that said root has said child, for modifying said text to be equal to said text added to said noun; and

second text return means for returning said text to said fact tree description means.

47. Method for describing, in a natural language, a query to a database previously implemented on a general purpose programmable digital computer, said computer comprising central processing means, bus means, display means, data entry means, memory means, data storage means, graphical user interface and

repository means further comprising relational database means implemented on said computer, said description method further comprising the steps of:

producing a diagram on said display means;

moving a cursor over said diagram;

forming a fact tree based on said query using a fact tree formation method for; and

describing said fact tree in said natural language using a fact tree description method.

48. The method of Claim 47, wherein said fact tree description method further comprises the steps of:

assigning variables based on said fact tree, said variables comprising root, parent, child and node, wherein said root is the root of said fact tree, said parent is the parent of said root, said child is the child of the root and said node is the number of the child using a first variable assignment method;

testing if a value of said parent is null using a parent test method;

creating text for said root using a root text creation method, responsive to a determination by said parent test method that said value of said parent is null;

creating text for said node using a node text creation method, responsive to a determination by said parent test method that a value of said parent is not null;

printing, on said display means, said text created by said root text creation method and said node text creation method using a print method;

counter method for counting an iteration as an iteration value;

determining if said iteration value is equal to a value of said node using a node test method; and

recursively invoking said fact tree description method using a depth-first recursive search method.

49. The method of Claim 48, said root text creation method further comprising the steps of:

assigning variables based on said fact tree, said variables comprising first noun, first restriction and first text, wherein said first noun is the noun in said

root, said first restriction is the restriction in said root and said first text is equal to the value of said first noun and a phrase indicating totality using a second variable assignment method;

testing if a value of said first restriction is null using a first restriction test method;

modifying said first text to be equal to said first text having and a word or phrase indicating presentation using a first text modification method, said first text modification method responsive to a determination by said first restriction test method that a value of said first restriction is not null;

modifying said text to be equal to said text and said noun and said restriction and a word or phrase indicating presentation using a second text modification method, said second text modification method responsive to a determination by said first restriction test method that a value of said restriction is null; and

returning said text as modified by said first or said second text modification method to said fact tree description method using a first text return method.

50. The method of Claim 48, wherein said node text creation method further comprises the steps of:

assigning variables based on said fact tree, said variables comprising second noun, parent-noun, phrase, second restriction and second text wherein said second noun is a noun in said root, said parent-noun is the noun in root' s parent, said phrase is the phrase of the root, said second restriction is a restriction in said root and said second text is equal to said second noun added to said parent noun added to said phrase, said method of assigning variables using a third variable assignment method;

testing if a value of said restriction is null using a second restriction test method;

modifying said text, responsive to a determination by said second restriction test method that a value of said restriction is not null, to be equal to said text added to said noun added to said restriction, using a third text modification method;

testing if said root has said children using a child testing method;

modifying said text, responsive to a determination by said child testing method that said root has said children; and

second text return method for returning said text to said fact tree description method.

51. Apparatus for creating a query to a database previously implemented on a general purpose programmable digital computer, said computer having central processing means, bus means, display means, data entry means, memory means, data storage means, and graphical user interface, said apparatus further comprising;

diagram means for producing a diagram on said display means;

cursor control means for moving a cursor over said diagram;

repository means further comprising relational database means implemented on said computer;

fact tree formation means for forming a fact tree, said fact tree further comprising a root node; and

query mapping means for mapping a query to said database means using said root node as an input and returning a first query, in a computer language appropriate to said database, equivalent to said root node.

52. The apparatus of Claim 51 further comprising:

said fact tree means with a non-root node having a relational mapping means associated therewith, said relational mapping means specifying said non-root node's representation in said database means; and

said query mapping means using said root non-root node as an input and returning a second query, in a computer language appropriate to said database means, equivalent to said non-root node.

53. The query mapping means of Claim 52 wherein said query mapping means further comprises:

query variable initialization means, responsive to a call to said query mapping means, for initializing query variables, said query variables comprising Root, WhereClause, FromList and SelectList, JoinTable and JoinColumn; said Root variable initialized to the value of root, and said WhereClause, FromList and SelectList, JoinTable and JoinColumn variables being initialized to a null value;

query build means, responsive to said query initialization means, for receiving said Root, JoinTable and JoinColumn variables, and building said query;

query variable format means format means for formatting said query variables after said query variables are returned from said query build means;

query assembly means for assembling said query from said query variables; and

query return means for returning said query mapping means.

54. The apparatus of Claim 53 wherein said query build means further comprises:

local build variable initialization means, responsive to invocation of said query build means for initializing local build variables, said local build variables comprising Root, Parent, Nodes and Child(i..nodes), Table and Column where said Root variable is set to the value of root, said Parent variable is set to the value of the parent of said root, said Nodes variable is the number of children of said root, Child(i..n) are children of said root, said Table variable is the table from said Child(i..n)'s mapping and said Column is the column for a second noun from said Child(i..n)'s mapping;

nodes checking means, responsive to said value of said Nodes variable, for returning to said query mapping means if said nodes checking means determines the value of said Nodes variable to be equal to zero;

parent checking means, responsive to a determination by said nodes checking means that the value of said nodes variable is greater than zero, for checking if the value of said Parent variable is a null value;

first selector adding means, responsive to a determination by said parent checking means that said value of said Parent variable is a null value, for updating said SelectList variable by adding to said SelectList variable the variables Table and Column, said Table variable equal to a table from said node's mapping and said Column variable equal to a column from said node's mapping, where said value of said node is equal to Child(i);

join creation means, responsive to a determination by said parent checking means that said value of said Parent variable is other than a null value, for joining a subquery to said query by adding an inner-join to said WhereClause;

iteration counter means for counting iterations;

iteration counter initialization means for initializing said iteration counter means to a value of 1;

iterative node test means for testing whether the value of said Nodes variable is less than or equal to the value of said iteration counter means, and returning to said query mapping means if said iterative node testing means determines the value of said Nodes variable to be greater than the value of said iteration counter;

second selector adding means, responsive to a determination by said iterative node test means that said Nodes variable is less than or equal to the value of said iteration counter means, for adding node's Table to said FromList variable and adding said node's table and column to said SelectList variable, where said value of said node is equal to Child(i), and Child(i) is the value of said iteration counter means;

recursive query build invoking means for joining queries created for child nodes of said node to said query;

iteration counter incrementing means for incrementing said iteration counter means by a value of 1; and

loop means for returning to said iterative node test means.

55. The apparatus of Claim 54 wherein said first selector adding means further comprises:

first variable initialization means for setting the value of local variable Node equal to the value of said Child(i) variable;

first table locating means for setting the value of said Table variable to the table of said Node variable's mapping;

first column locating means for setting the value of said Column variable to the column for the first noun from said Node variable's mapping;

first listing means for setting the value of said SelectList variable to the previous value of said SelectList variable added to said Tabie variable and said Column variable; and

first selector return means for returning said value of said SelectList variable to said query build means.

56. The apparatus of Claim 54 wherein said join creation means further comprises:

second variable initialization means for setting the value of local variable Node equal to the value of said Child(i) variable;

second table locating means for setting the value of said Table variable to the table of said Node variable's mapping;

second column locating means for setting the value of said Column variable to the column for the first noun from said Node variable's mapping;

join assembly means for assembling said join by setting the value of said WhereClause variable to the value of said WhereClause variable added to said Table variable added to said Column variable added to said JoinTable variable added to said JoinColumn variable; and

join return means for returning said join assembled by said join assembly means to said query build means.

57. The apparatus of Claim 54 wherein said second selector adding means further comprises third variable initialization means for setting the value of local variable Node equal to the value of said Child(i) variable which is equal to the value of said iteration counter means;

third table locating means for setting the value of said Table variable to the table of said Node variable's mapping;

third column locating means for setting the value of said Column variable to the column for the first noun from said Node variable's mapping;

second listing means for setting the value of said SelectList variable to the previous value of said SelectList variable added to said Table variable and said Column variable;

third listing means for setting the value of said FromList variable to the previous value of said FromList variable added to said Table variable; and

second selector return means for returning said value of said FromList variable and said SelectList variable to said query build means.

58. Method for creating a query to a database previously implemented on a general purpose programmable digital computer, said computer having central processing means, bus means, display means, data entry means, memory means,

data storage means, and graphical user interface, said method further comprising the steps of;

producing a diagram on said display means using a diagram means;

moving a cursor over said diagram using a cursor control means;

implementing on said computer a repository means further comprising relational database means;

forming a fact tree on a fact tree formation means, said fact tree further comprising a root node; and

mapping a query to said database means using a query mapping means and using said root node as an input and returning a first query, in a computer language appropriate to said database, equivalent to said root node.

59. The method of Claim 58 further comprising the steps of:

specifying in said database means a representation of a non-root node of said fact tree said non-root node having a relational mapping means associated therewith; and

returning a second query, in a computer language appropriate to said database means, equivalent to said non-root node using said query mapping means and using said root non-root node as an input.

60. The method of Claim 59 wherein said query mapping means further comprises the steps of:

initializing query variables, said query variables comprising Root, WhereClause, FromList and SelectList, JoinTable and JoinColumn; said Root variable initialized to the value of root, and said WhereClause, FromList and SelectList, JoinTable and JoinColumn variables being initialized to a null value using query variable initialization means, responsive to a call to said query mapping means;

building said query using query build means responsive to said Root, JoinTable and JoinColumn variables, and to said query initialization means;

formatting said query variables using query variable format means after said query variables are returned from said query build means;

assembling said query from said query variables using query assembly means; and

returning said query mapping means using query return means.

61. The method of Claim 60 wherein said query build means further comprises the steps of:

initializing local build variables using local build variable initialization means responsive to invocation of said query build means, said local build variables comprising Root, Parent, Nodes and Child(i..nodes), Table and Column where said Root variable is set to the value of root, said Parent variable is set to the value of the parent of said root, said Nodes variable is the number of children of said root, Child(i..n) are children of said root, said Table variable is the table from said Child(i..n)'s mapping and said Column is the column for a second noun from said Child(i..n)'s mapping;

returning to said query mapping means if said nodes checking means determines the value of said Nodes variable to be equal to zero, using nodes checking means, responsive to said value of said Nodes variable;

checking if the value of said Parent variable is a null value using parent checking means, responsive to a determination by said nodes checking means that the value of said nodes variable is greater than zero;

updating said SelectList variable by adding to said SelectList variable the variables Table and Column, said Table variable equal to a table from said node's mapping and said Column variable equal to a column from said node's mapping, where said value of said node is equal to Child(i) using first selector adding means, responsive to a determination by said parent checking means that said value of said Parent variable is a null value;

joining a subquery to said query by adding an inner-join to said WhereClause using join creation means, responsive to a determination by said parent checking means that said value of said Parent variable is other than a null value;

counting iterations using iteration counter means;

initializing said iteration counter means to a value of 1 using iteration counter initialization means;

testing whether the value of said Nodes variable is less than or equal to the value of said iteration counter means, and returning to said query mapping means if said iterative node testing means determines the value of said Nodes

variable to be greater than the value of said iteration counter means using iterative node test means;

adding node's table to said FromList variable and adding said node's table and column to said SelectList variable, where said value of said node is equal to Child(i), and Child(i) is the value of said iteration counter means using second selector adding means, responsive to a determination by said iterative node test means that said Nodes variable is less than or equal to the value of said iteration counter means;

joining queries created for child nodes of said node to said query using recursive query build invoking means;

incrementing said iteration counter means by a value of 1 using iteration counter incrementing means; and

returning to said iterative node test means using loop means.

62. The method of Claim 61 wherein said first selector adding means further comprises the steps of:

setting the value of local variable Node equal to the value of said Child(i) variable using first variable initialization means;

setting the value of said Table variable to the table of said Node variable's mapping using first table locating means;

setting the value of said Column variable to the column for the first noun from said Node variable's mapping using first column locating means;

setting the value of said SelectList variable to the previous value of said SelectList variable added to said Table variable and said Column variable using first listing means; and

returning said value of said SelectList variable to said query build means using first selector return means.

63. The method of Claim 62 wherein said join creation means further comprises the steps of:

setting the value of local variable Node equal to the value of said Child(i) variable using second variable initialization means;

setting the value of said Table variable to the table of said Node variable's mapping using second table locating means;

setting the value of said Column variable to the column for the first noun from said Node variable's mapping using second column locating means;

assembling said join by setting the value of said WhereClause variable to the value of said WhereClause variable added to said Table variable added to said Column variable added to said JoinTable variable added to said JoinColumn variable using join assembly means; and

returning said join assembled by said join assembly means to said query build means using join return means.

64. The method of Claim 61 wherein said second selector adding means further comprises the steps of third variable initialization means for setting the value of local variable Node equal to the value of said Child(i) variable which is equal to the value of said iteration counter means;

setting the value of said Table variable to the table of said Node variable's mapping using third table locating means;

setting the value of said Column variable to the column for the first noun from said Node variable's mapping using third column locating means;

setting the value of said SelectList variable to the previous value of said SelectList variable added to said Table variable and said Column variable using second listing means;

setting the value of said FromList variable to the previous value of said FromList variable added to said Table variable using third listing means; and

returning said value of said FromList variable and said SelectList variable to said query build means using second selector return means.

65. Apparatus including a general purpose programmable digital computer, said computer having central processing means, bus means, display means, data entry means, memory means, data storage means, and graphical user interface, for specifying database designs, creating a query to said database and for describing said query in a natural language, said apparatus comprising;

diagram means for producing a diagram on said display means;

cursor control means for controlling movement of a cursor over said diagram;

repository means further comprising relational database means implemented on said computer;

text input means for entering, into an edit window, text using said data entry means, items of said text including data objects, facts about said data objects and constraints on said data objects;

user-selectable text validation means for validating said text;

user-selectable translation means for translating said text from said text input means into said diagram;

user-selectable first compilation means for compiling said text only into said repository means;

fact tree formation means for forming a fact tree based on said query;

fact tree description means for describing said fact tree in said natural language;

said fact tree means further including a root node; and

query mapping means for mapping a query to said database means using said root node as an input and returning a first query, in a computer language appropriate to said database, equivalent to said root node.

66. Method for specifying database designs, for creating a query to said database and for describing said query in a natural language on an apparatus including a general purpose programmable digital computer, said computer having central processing means, bus means, display means, data entry means, memory means, data storage means, and graphical user interface, said method comprising the steps of;

producing a diagram on said display means using diagram means;

controlling movement of a cursor over said diagram using cursor control means;

repository means further comprising relational database means implemented on said computer;

using said data entry means to enter into an edit window items of said text including data objects, facts about said data objects and constraints on said data objects;

validating said text using user-selectable text validation means;

translating said text from said text input means into said diagram using user-selectable translation means;

compiling said text only into said repository means using user-selectable first compilation means;

forming a fact tree based on said query using fact tree formation means;

describing said fact tree in a computer language approximating a natural language, using fact tree description means, said fact tree means further including a root node; and

mapping a query to said database means using query mapping means and said root node as an input and returning a first query, in a computer language appropriate to said database, equivalent to said root node.

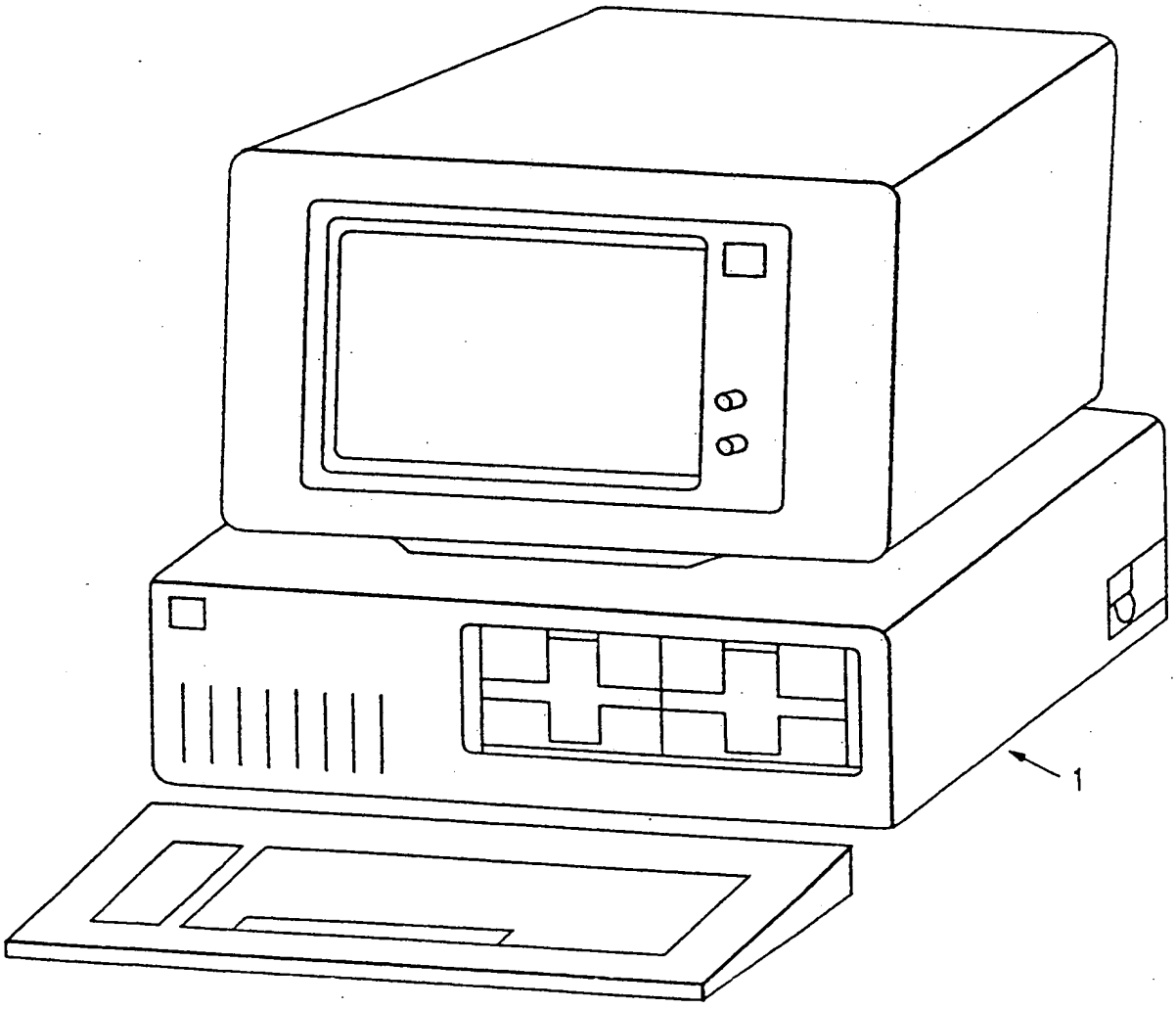


FIG. 1

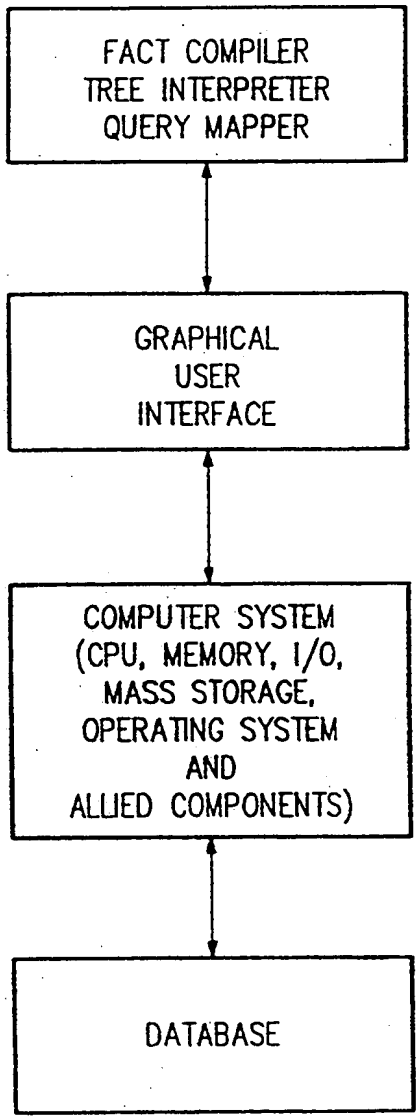

FACT COMPILER
TREE INTERPRETER
QUERY MAPPER
GRAPHICAL
USER
INTERFACE
COMPUTER SYSTEM
(CPU, MEMORY, I/O,
MASS STORAGE,
OPERATING SYSTEM
AND
ALLIED COMPONENTS)
DATABASE

FIG. 2

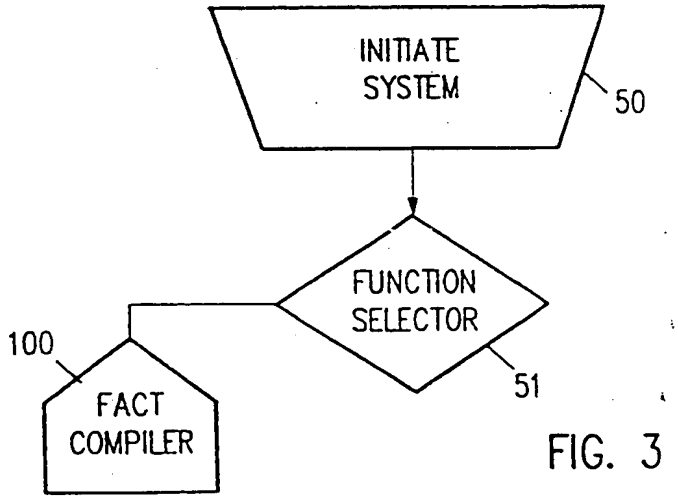

INITIATE
SYSTEM
50
FUNCTION
SELECTOR
51
100
FACT
COMPILER

FIG. 3

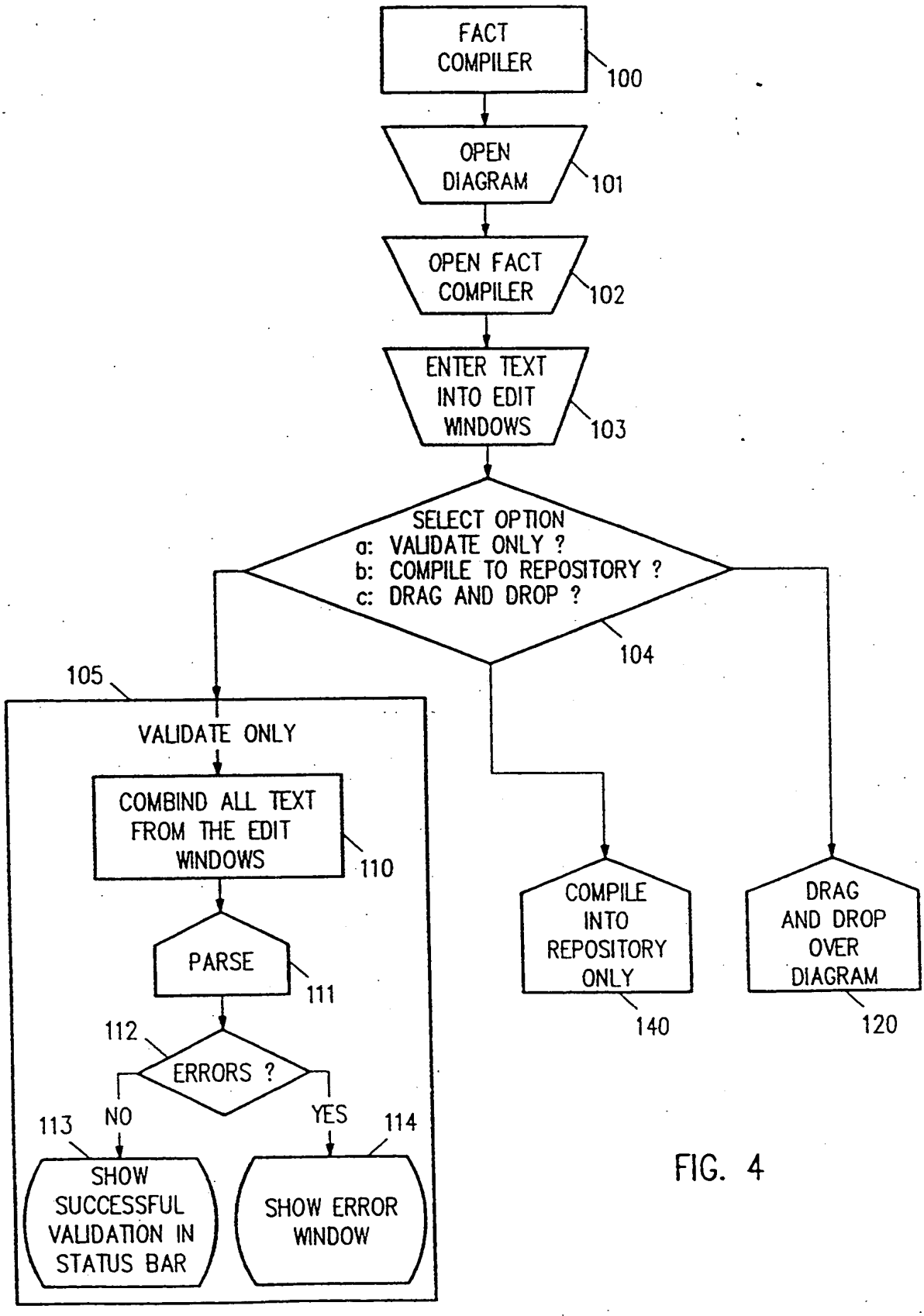

FACT COMPILER
100
OPEN DIAGRAM
101
OPEN FACT COMPILER
102
ENTER TEXT INTO EDIT WINDOWS
103
SELECT OPTION
a: VALIDATE ONLY ?
b: COMPILE TO REPOSITORY ?
c: DRAG AND DROP ?
104
105
VALIDATE ONLY
COMBIND ALL TEXT FROM THE EDIT WINDOWS
110
PARSE
111
112
ERRORS ?
NO
YES
113
114
SHOW SUCCESSFUL VALIDATION IN STATUS BAR
SHOW ERROR WINDOW
COMPILE INTO REPOSITORY ONLY
140
DRAG AND DROP OVER DIAGRAM
120

FIG. 4

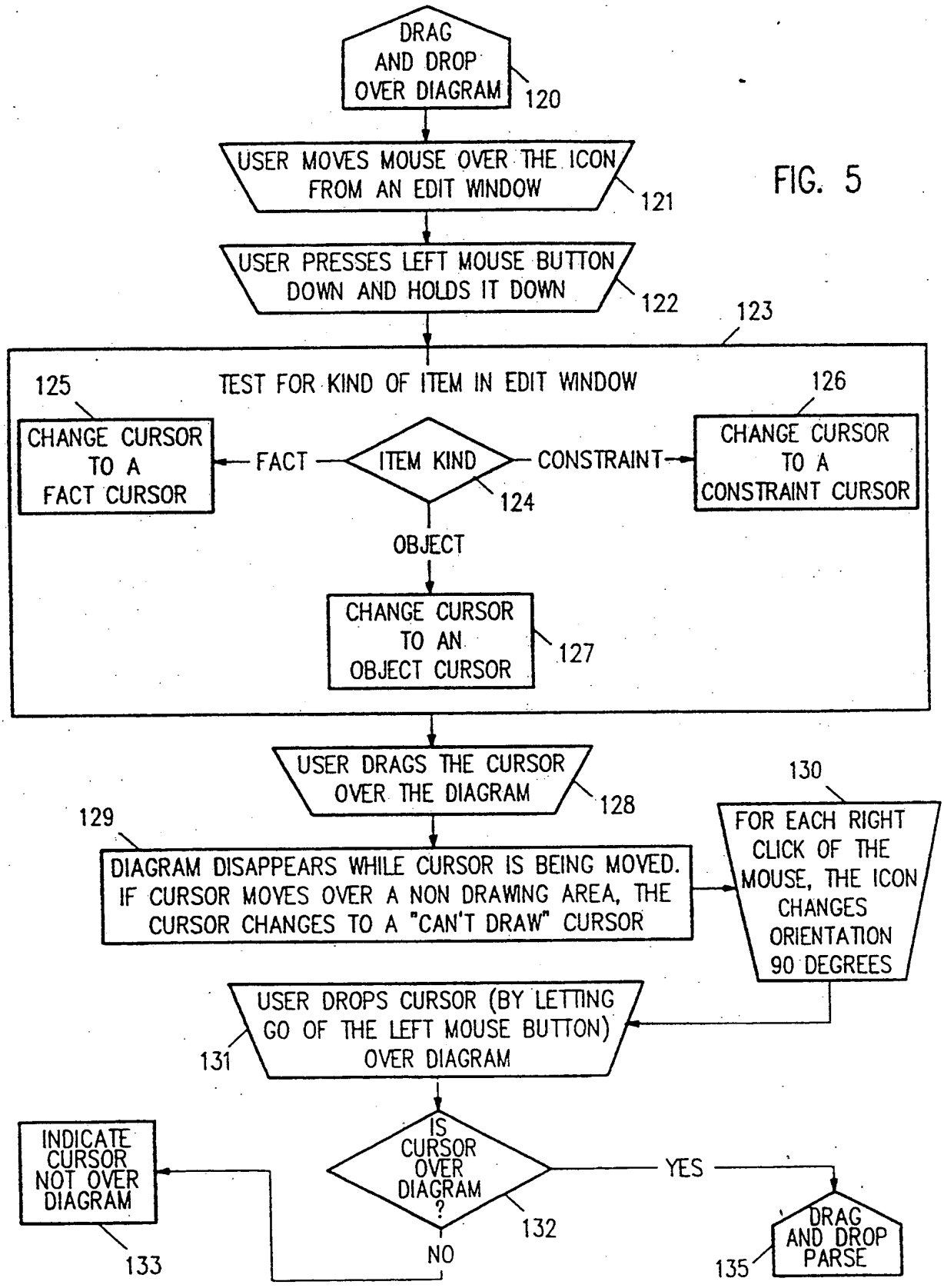

DRAG AND DROP OVER DIAGRAM
120
USER MOVES MOUSE OVER THE ICON FROM AN EDIT WINDOW
121
USER PRESSES LEFT MOUSE BUTTON DOWN AND HOLDS IT DOWN
122
123
TEST FOR KIND OF ITEM IN EDIT WINDOW
125
CHANGE CURSOR TO A FACT CURSOR
FACT
ITEM KIND
124
CONSTRAINT
126
CHANGE CURSOR TO A CONSTRAINT CURSOR
OBJECT
CHANGE CURSOR TO AN OBJECT CURSOR
127
USER DRAGS THE CURSOR OVER THE DIAGRAM
128
129
DIAGRAM DISAPPEARS WHILE CURSOR IS BEING MOVED. IF CURSOR MOVES OVER A NON DRAWING AREA, THE CURSOR CHANGES TO A "CAN'T DRAW" CURSOR
130
FOR EACH RIGHT CLICK OF THE MOUSE, THE ICON CHANGES ORIENTATION 90 DEGREES
USER DROPS CURSOR (BY LETTING GO OF THE LEFT MOUSE BUTTON) OVER DIAGRAM
131
IS CURSOR OVER DIAGRAM ?
132
YES
NO
INDICATE CURSOR NOT OVER DIAGRAM
133
DRAG AND DROP PARSE
135

FIG. 5

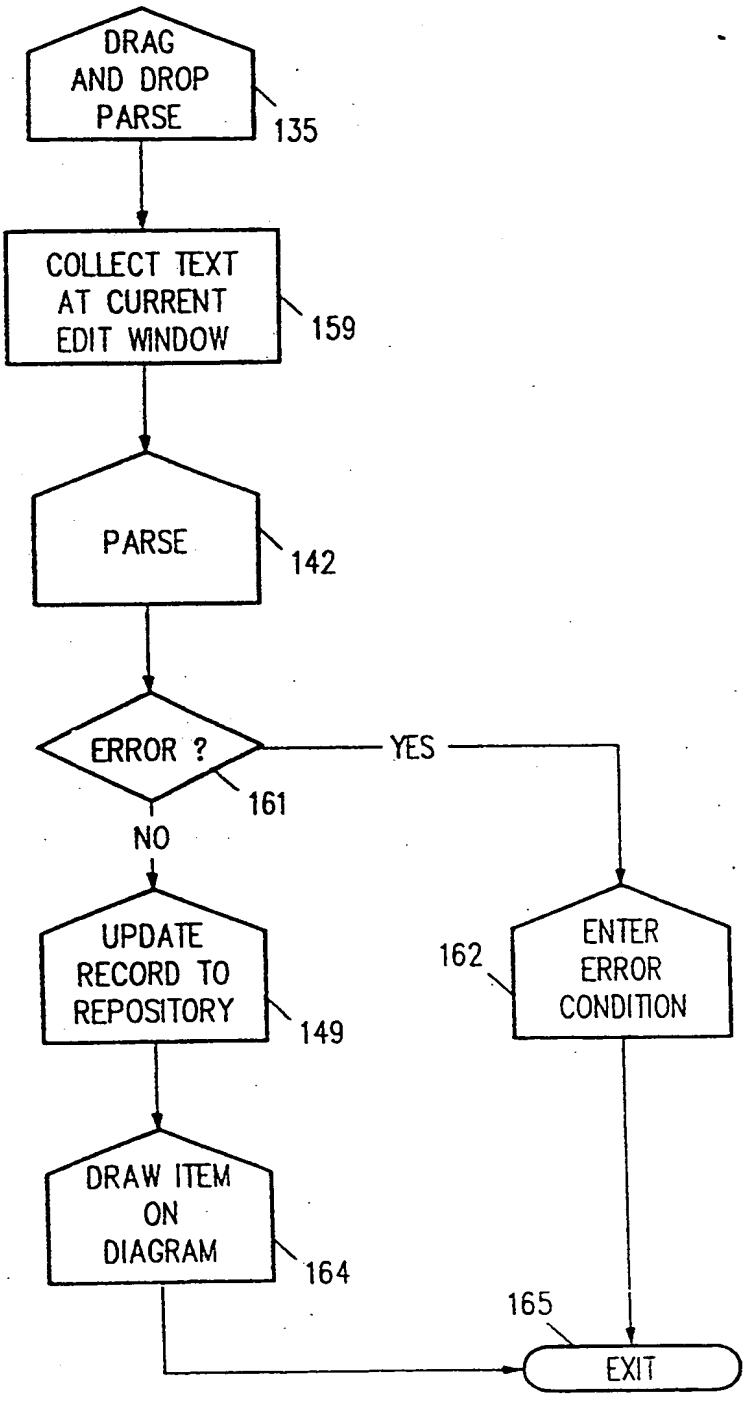

DRAG AND DROP PARSE
135
COLLECT TEXT AT CURRENT EDIT WINDOW
159
PARSE
142
ERROR ?
161
YES
NO
UPDATE RECORD TO REPOSITORY
149
ENTER ERROR CONDITION
162
DRAW ITEM ON DIAGRAM
164
165
EXIT

FIG. 6

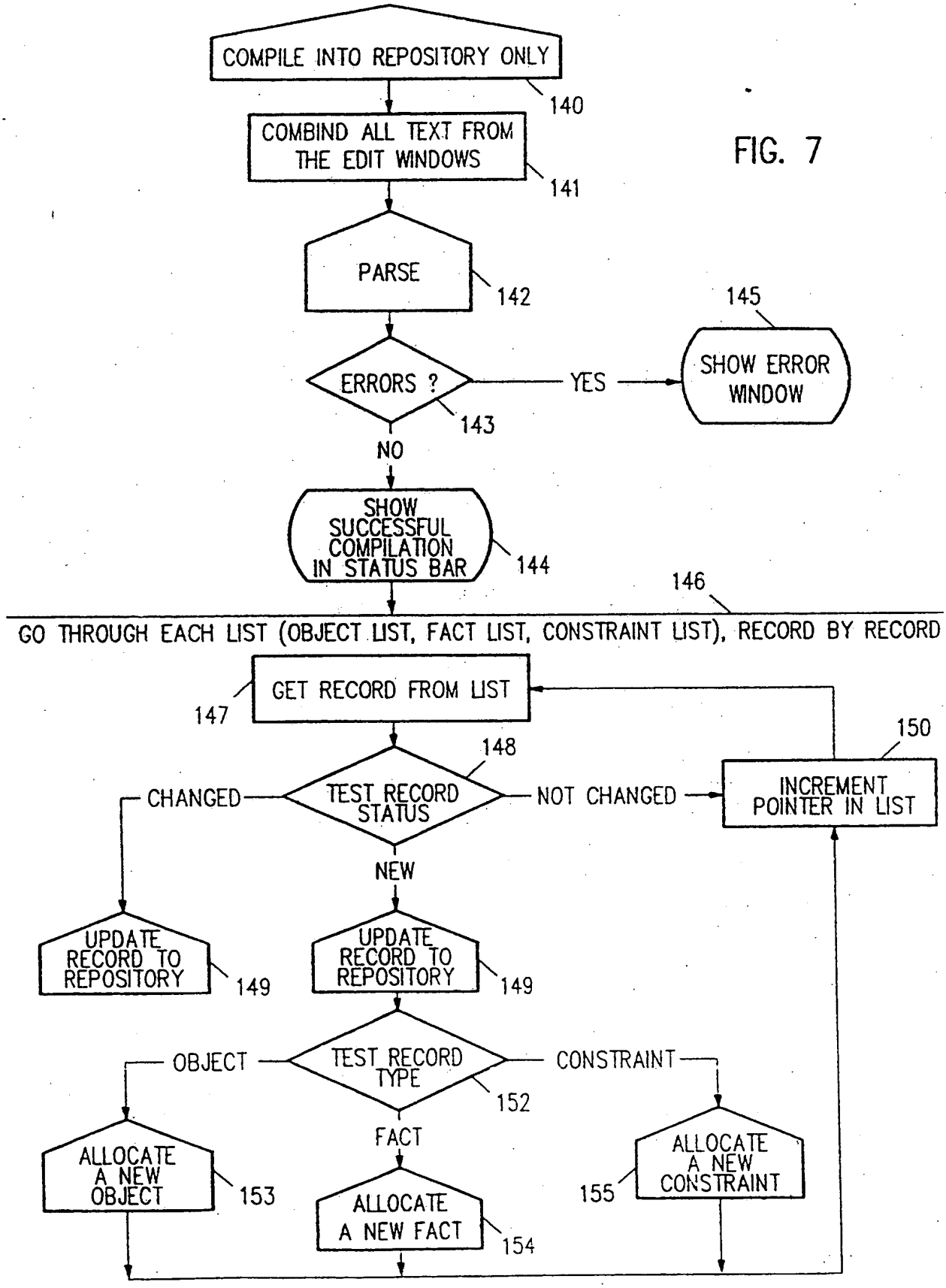

COMPILE INTO REPOSITORY ONLY
140
COMBIND ALL TEXT FROM THE EDIT WINDOWS
141
PARSE
142
145
ERRORS ?
YES
SHOW ERROR WINDOW
143
NO
SHOW SUCCESSFUL COMPILATION IN STATUS BAR
144
146
GO THROUGH EACH LIST (OBJECT LIST, FACT LIST, CONSTRAINT LIST), RECORD BY RECORD
GET RECORD FROM LIST
147
148
150
CHANGED
TEST RECORD STATUS
NOT CHANGED
INCREMENT POINTER IN LIST
NEW
UPDATE RECORD TO REPOSITORY
149
UPDATE RECORD TO REPOSITORY
149
OBJECT
TEST RECORD TYPE
CONSTRAINT
152
FACT
ALLOCATE A NEW OBJECT
153
ALLOCATE A NEW FACT
154
155
ALLOCATE A NEW CONSTRAINT

FIG. 7

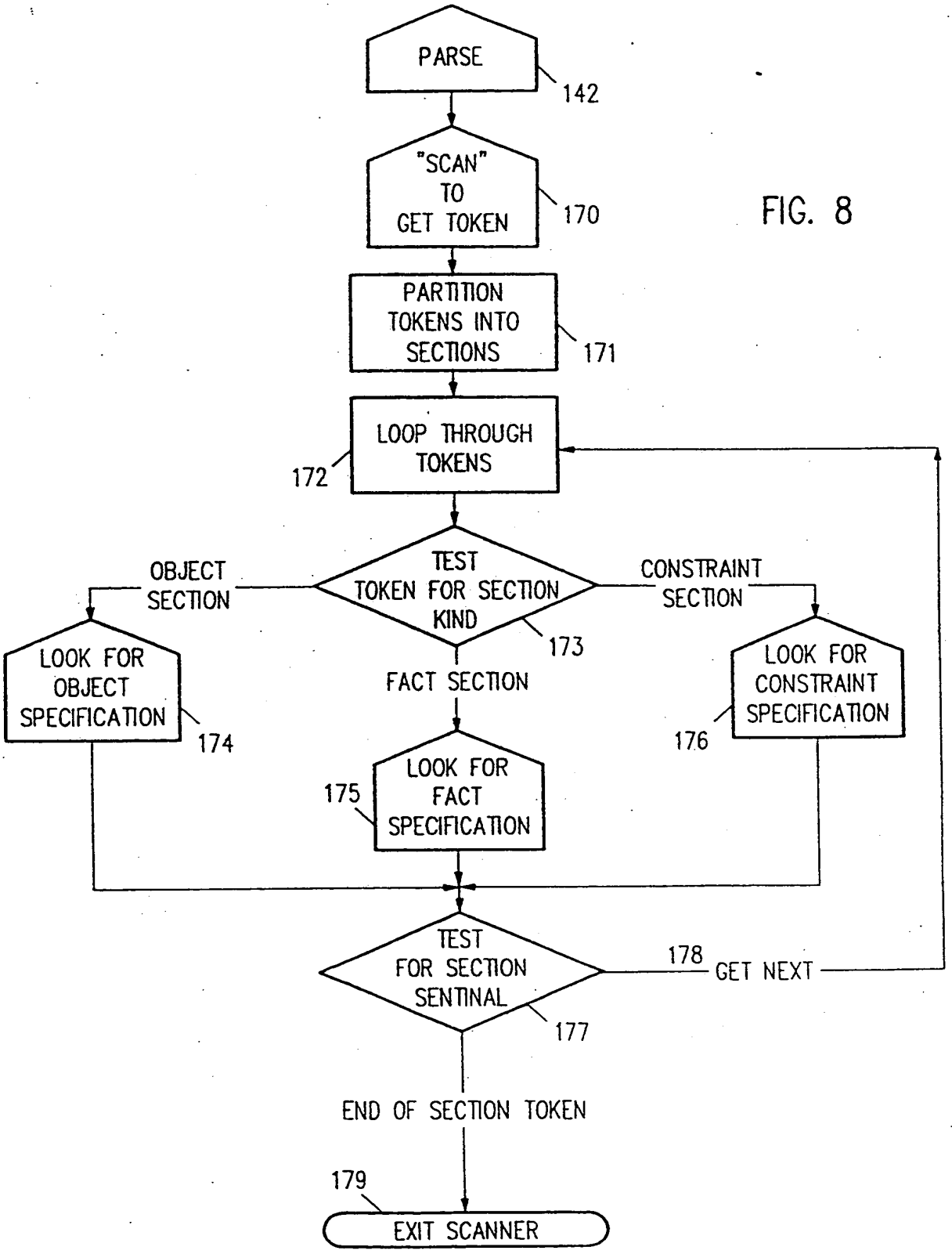

PARSE
142
"SCAN" TO GET TOKEN
170
PARTITION TOKENS INTO SECTIONS
171
LOOP THROUGH TOKENS
172
TEST TOKEN FOR SECTION KIND
173
OBJECT SECTION
FACT SECTION
CONSTRAINT SECTION
LOOK FOR OBJECT SPECIFICATION
174
LOOK FOR FACT SPECIFICATION
175
LOOK FOR CONSTRAINT SPECIFICATION
176
TEST FOR SECTION SENTINAL
177
178
GET NEXT
END OF SECTION TOKEN
179
EXIT SCANNER

FIG. 8

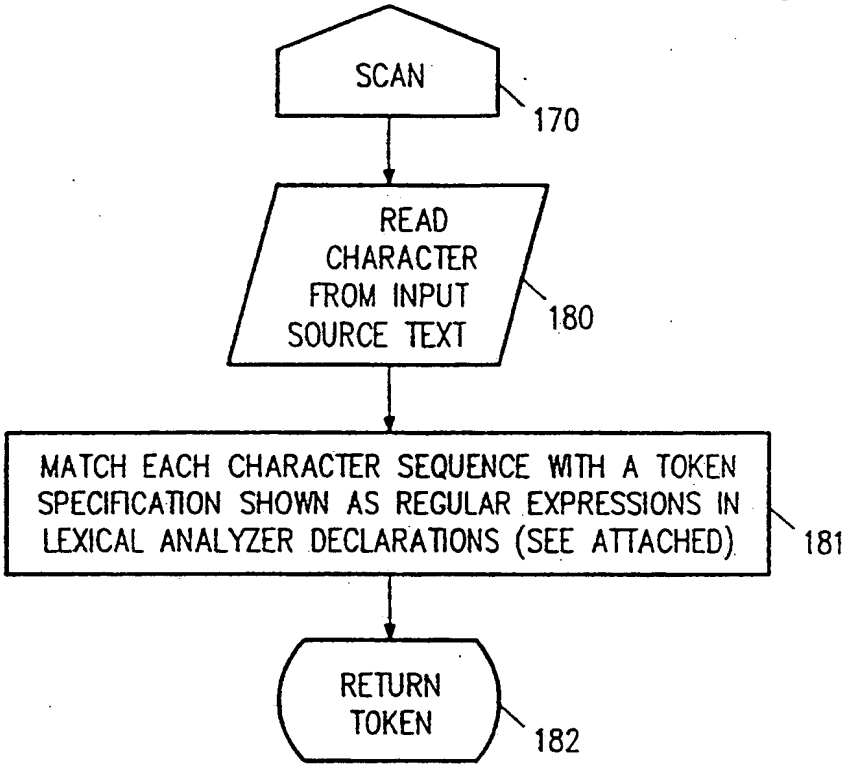

SCAN
170
READ CHARACTER FROM INPUT SOURCE TEXT
180
MATCH EACH CHARACTER SEQUENCE WITH A TOKEN SPECIFICATION SHOWN AS REGULAR EXPRESSIONS IN LEXICAL ANALYZER DECLARATIONS (SEE ATTACHED)
181
RETURN TOKEN
182

FIG. 9

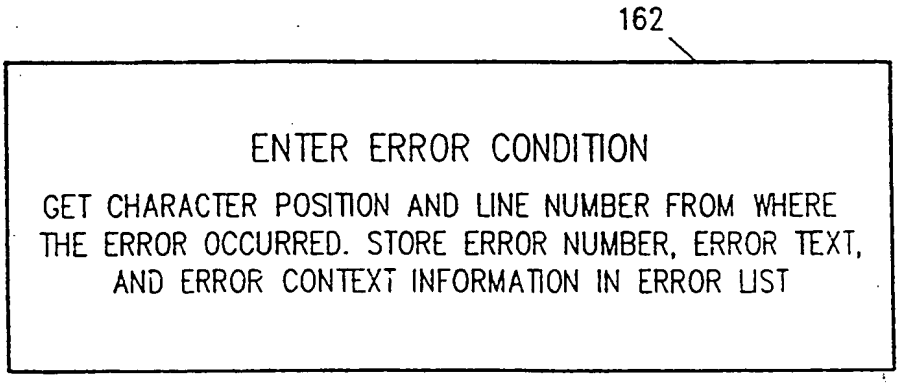

162
ENTER ERROR CONDITION
GET CHARACTER POSITION AND LINE NUMBER FROM WHERE THE ERROR OCCURRED. STORE ERROR NUMBER, ERROR TEXT, AND ERROR CONTEXT INFORMATION IN ERROR LIST

FIG. 10

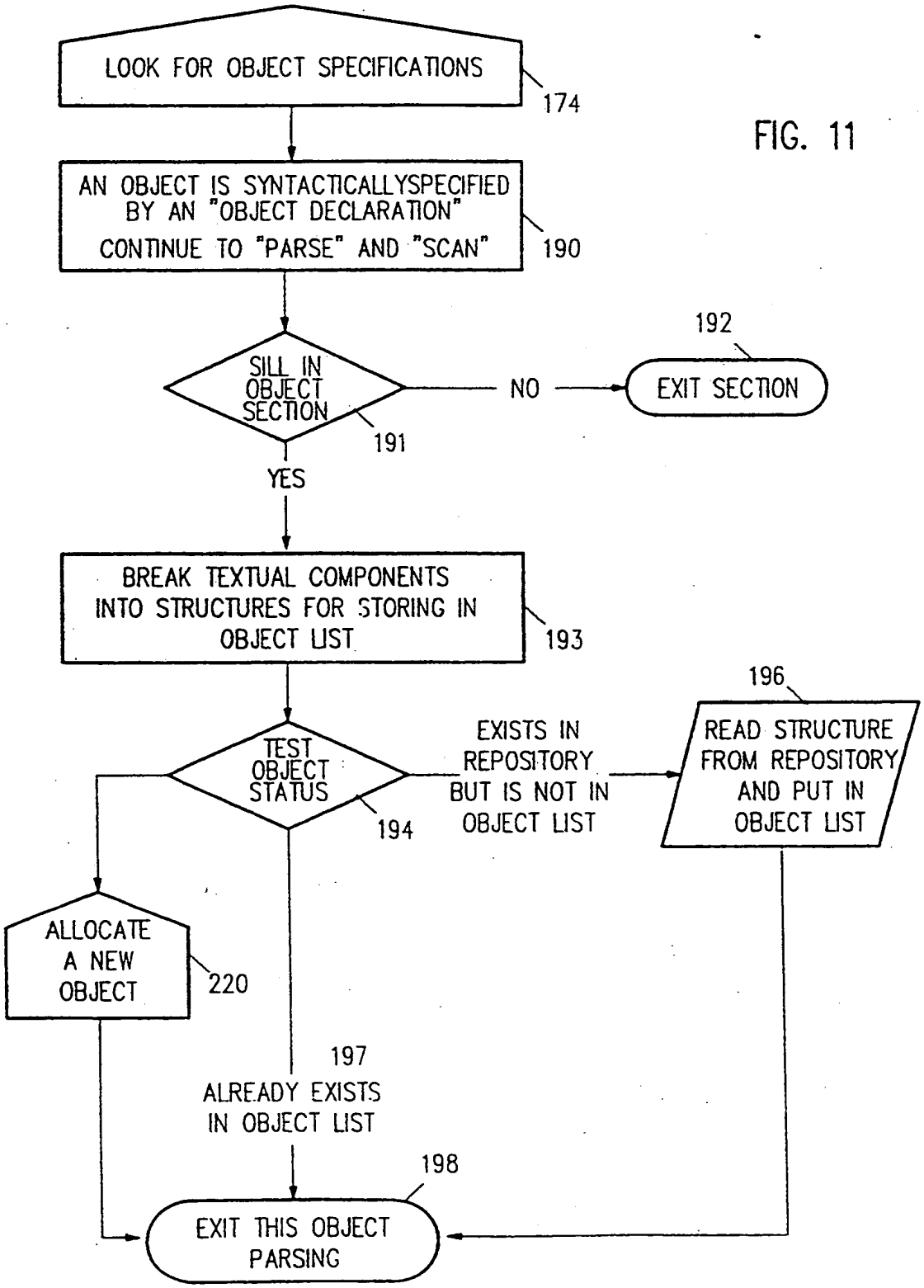

LOOK FOR OBJECT SPECIFICATIONS
174
AN OBJECT IS SYNTACTICALLYSPECIFIED BY AN "OBJECT DECLARATION"
CONTINUE TO "PARSE" AND "SCAN"
190
192
SILL IN OBJECT SECTION
191
NO
EXIT SECTION
YES
BREAK TEXTUAL COMPONENTS INTO STRUCTURES FOR STORING IN OBJECT LIST
193
196
TEST OBJECT STATUS
194
EXISTS IN REPOSITORY BUT IS NOT IN OBJECT LIST
READ STRUCTURE FROM REPOSITORY AND PUT IN OBJECT LIST
ALLOCATE A NEW OBJECT
220
197
ALREADY EXISTS IN OBJECT LIST
198
EXIT THIS OBJECT PARSING

FIG. 11

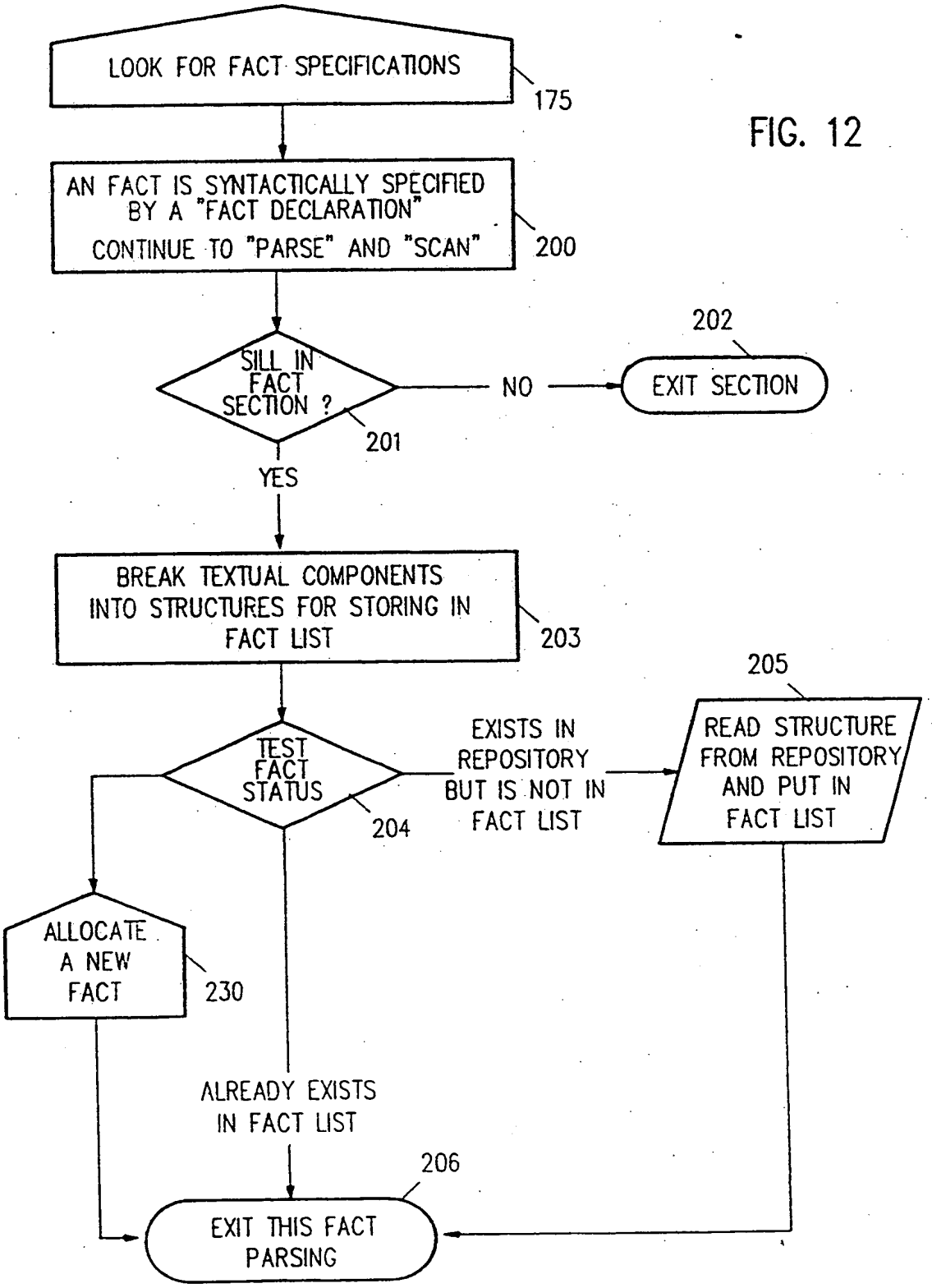

LOOK FOR FACT SPECIFICATIONS
175
AN FACT IS SYNTACTICALLY SPECIFIED BY A "FACT DECLARATION"
CONTINUE TO "PARSE" AND "SCAN"
200
SILL IN FACT SECTION ?
201
NO
202
EXIT SECTION
YES
BREAK TEXTUAL COMPONENTS INTO STRUCTURES FOR STORING IN FACT LIST
203
TEST FACT STATUS
204
EXISTS IN REPOSITORY BUT IS NOT IN FACT LIST
205
READ STRUCTURE FROM REPOSITORY AND PUT IN FACT LIST
ALLOCATE A NEW FACT
230
ALREADY EXISTS IN FACT LIST
206
EXIT THIS FACT PARSING

FIG. 12

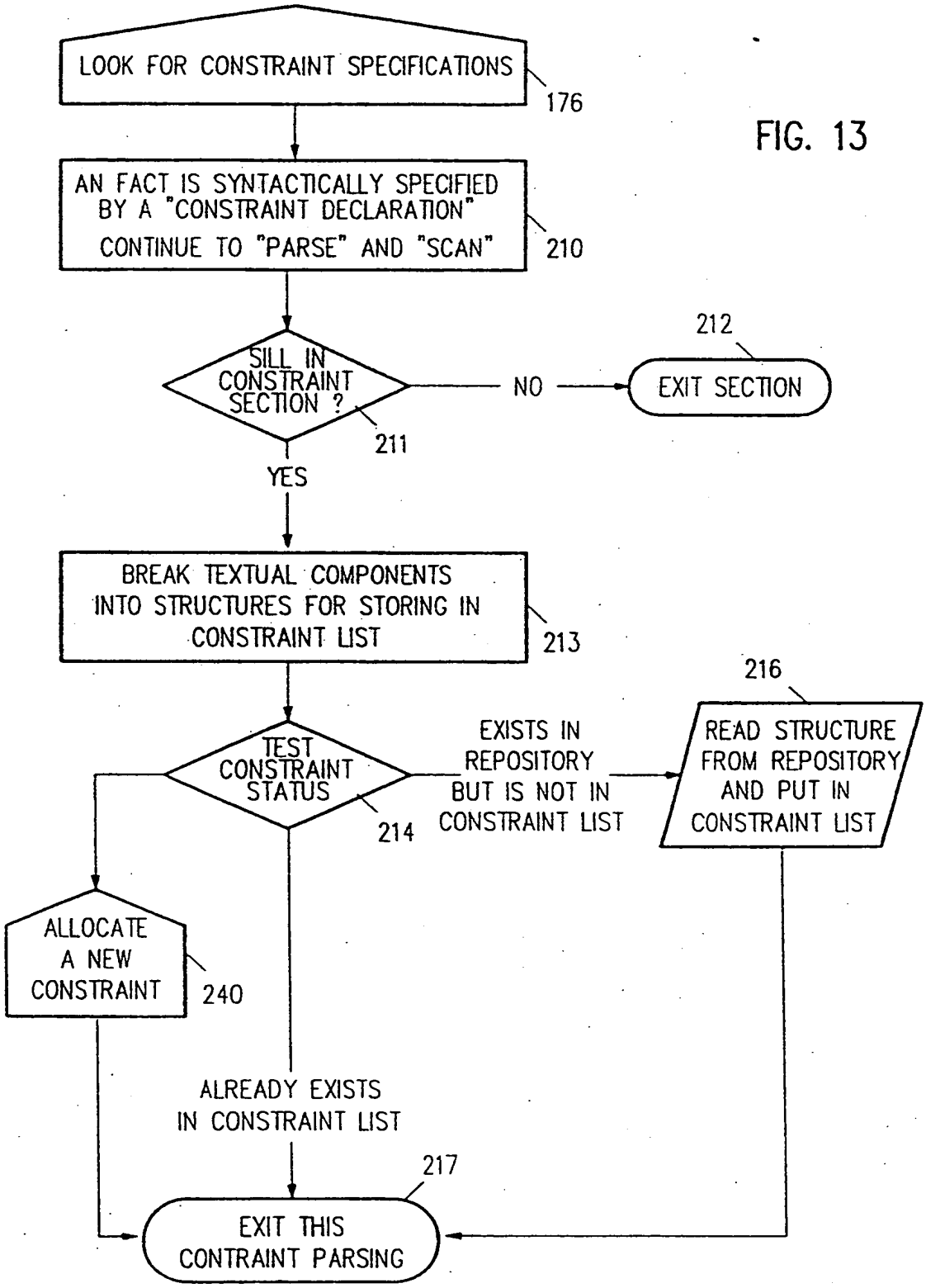

LOOK FOR CONSTRAINT SPECIFICATIONS
176
AN FACT IS SYNTACTICALLY SPECIFIED BY A "CONSTRAINT DECLARATION"
CONTINUE TO "PARSE" AND "SCAN"
210
SILL IN CONSTRAINT SECTION ?
211
NO
212
EXIT SECTION
YES
BREAK TEXTUAL COMPONENTS INTO STRUCTURES FOR STORING IN CONSTRAINT LIST
213
TEST CONSTRAINT STATUS
214
EXISTS IN REPOSITORY BUT IS NOT IN CONSTRAINT LIST
216
READ STRUCTURE FROM REPOSITORY AND PUT IN CONSTRAINT LIST
ALLOCATE A NEW CONSTRAINT
240
ALREADY EXISTS IN CONSTRAINT LIST
217
EXIT THIS CONTRAINT PARSING

FIG. 13

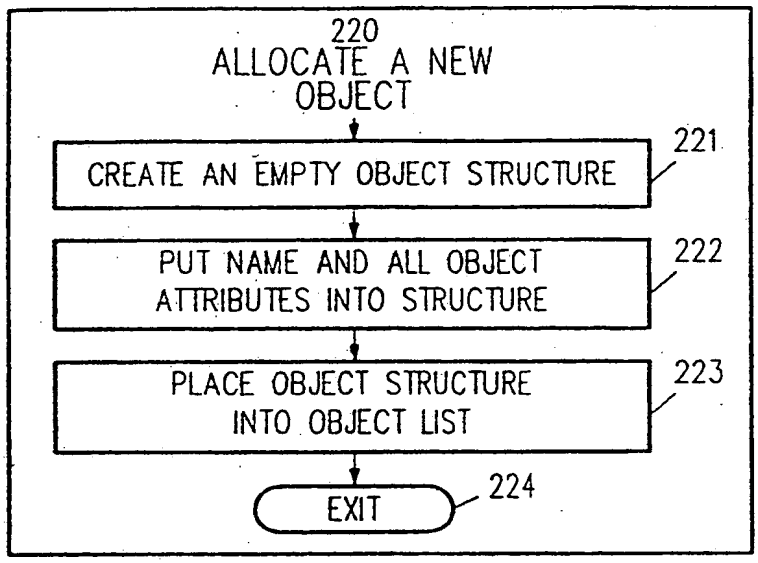


FIG. 14

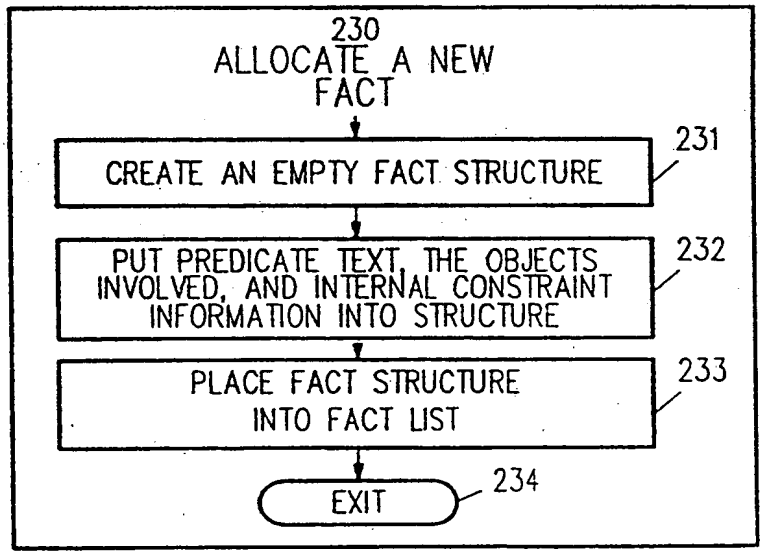


FIG. 15

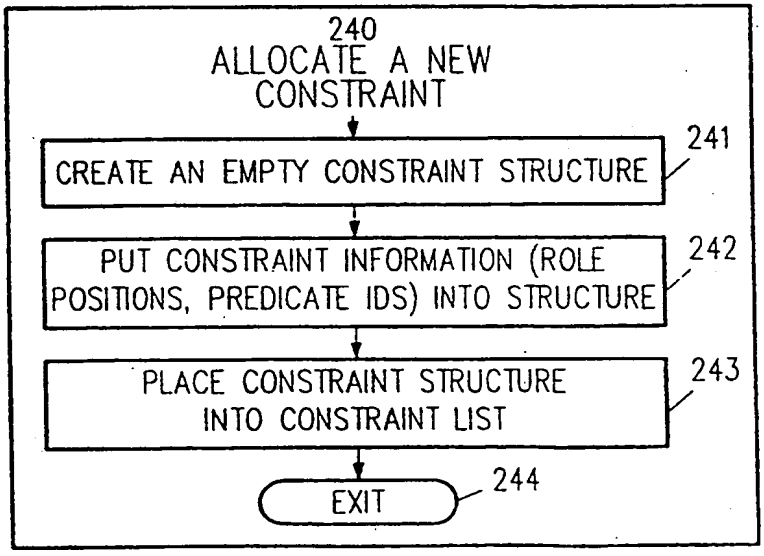


FIG. 16

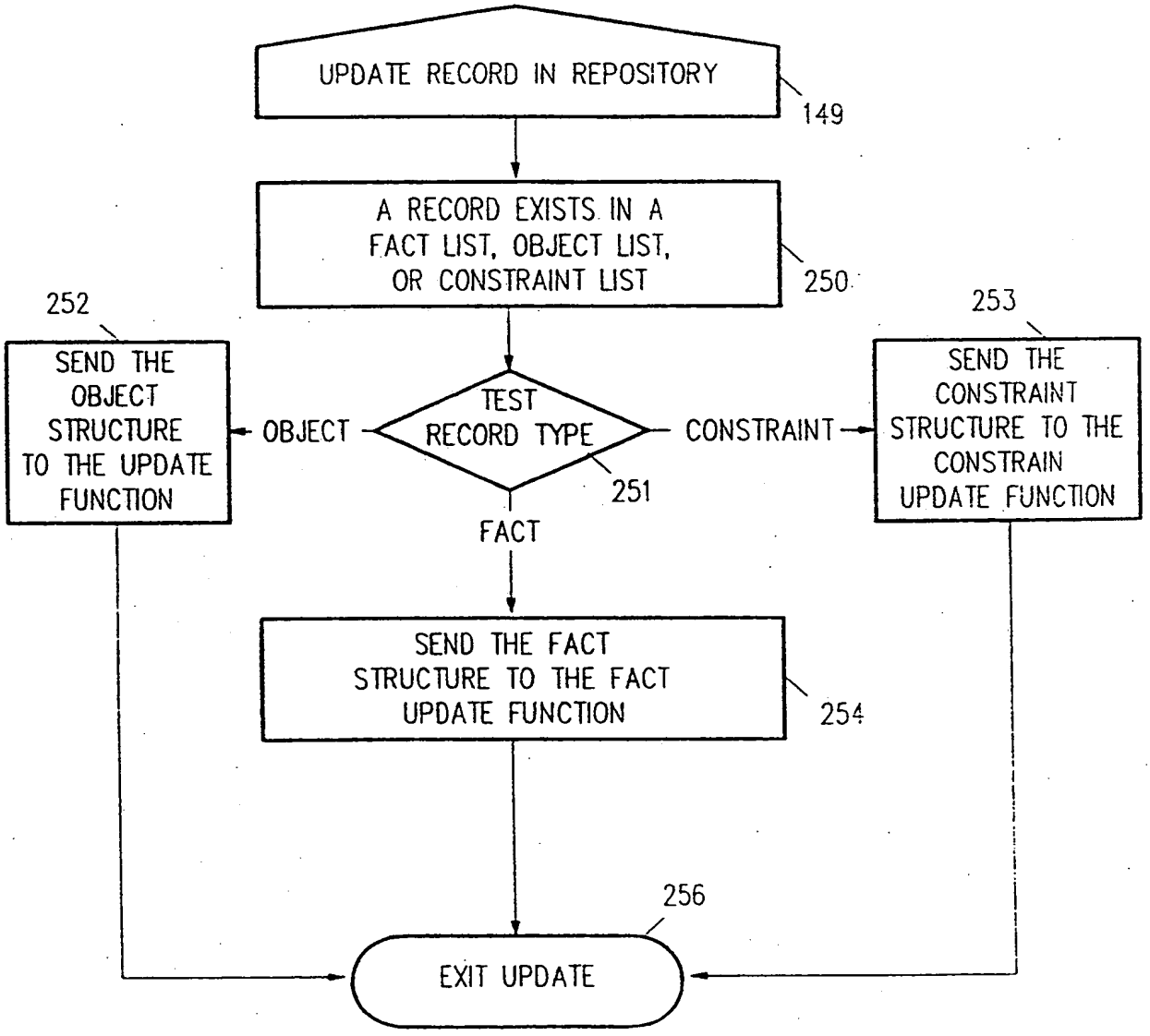
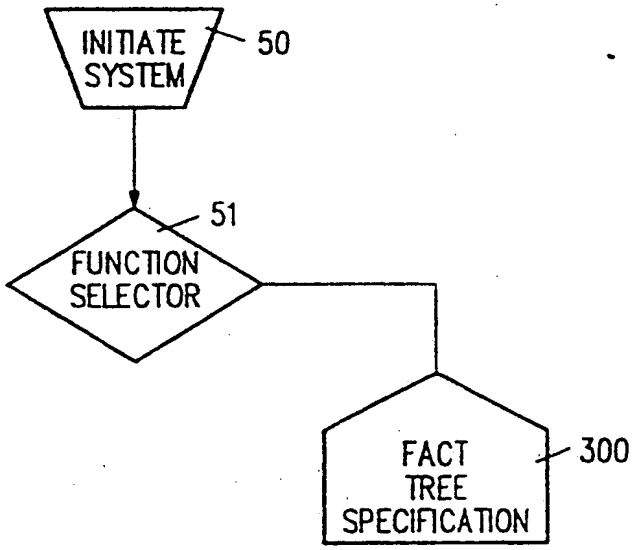

UPDATE RECORD IN REPOSITORY
149
A RECORD EXISTS IN A FACT LIST, OBJECT LIST, OR CONSTRAINT LIST
250
252
SEND THE OBJECT STRUCTURE TO THE UPDATE FUNCTION
OBJECT
TEST RECORD TYPE
251
CONSTRAINT
253
SEND THE CONSTRAINT STRUCTURE TO THE CONSTRAIN UPDATE FUNCTION
FACT
SEND THE FACT STRUCTURE TO THE FACT UPDATE FUNCTION
254
256
EXIT UPDATE

FIG. 17

FIG. 18


INITIATE SYSTEM
50
51
FUNCTION SELECTOR
FACT TREE SPECIFICATION
300

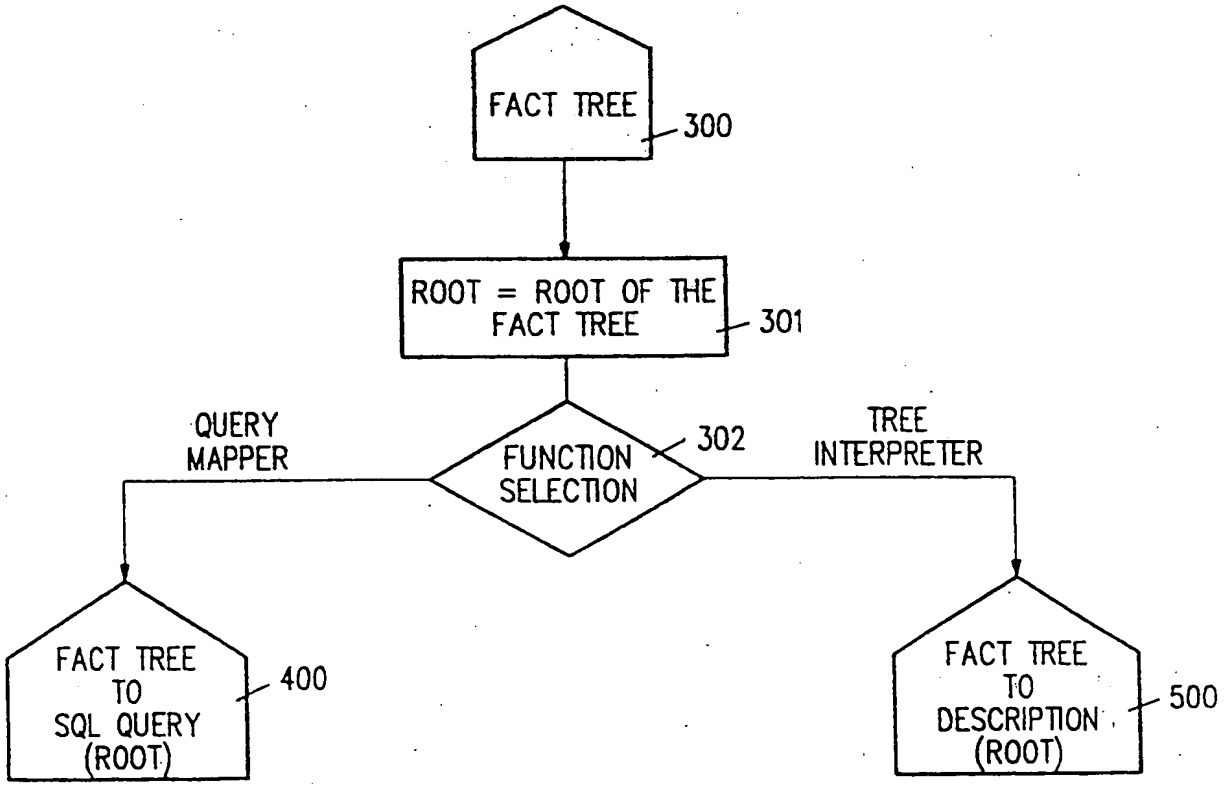

FACT TREE
300
ROOT = ROOT OF THE FACT TREE
301
QUERY MAPPER
FUNCTION SELECTION
302
TREE INTERPRETER
FACT TREE TO SQL QUERY (ROOT)
400
FACT TREE TO DESCRIPTION (ROOT)
500

FIG. 19

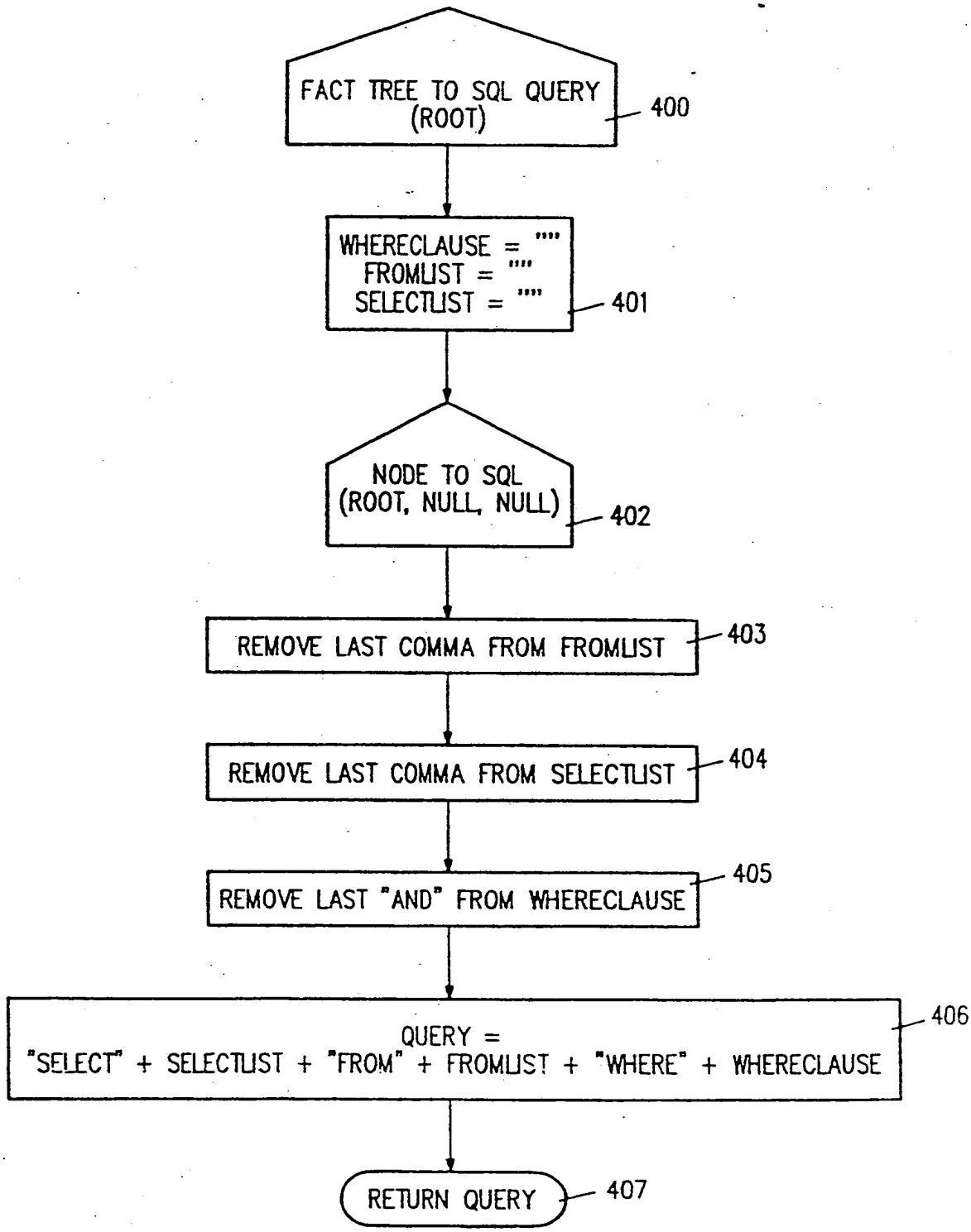

FACT TREE TO SQL QUERY (ROOT)
400
WHERECLAUSE = ""
FROMLIST = ""
SELECTLIST = ""
401
NODE TO SQL (ROOT, NULL, NULL)
402
REMOVE LAST COMMA FROM FROMLIST
403
REMOVE LAST COMMA FROM SELECTLIST
404
REMOVE LAST "AND" FROM WHERECLAUSE
405
QUERY = "SELECT" + SELECTLIST + "FROM" + FROMLIST + "WHERE" + WHERECLAUSE
406
RETURN QUERY
407

FIG. 20

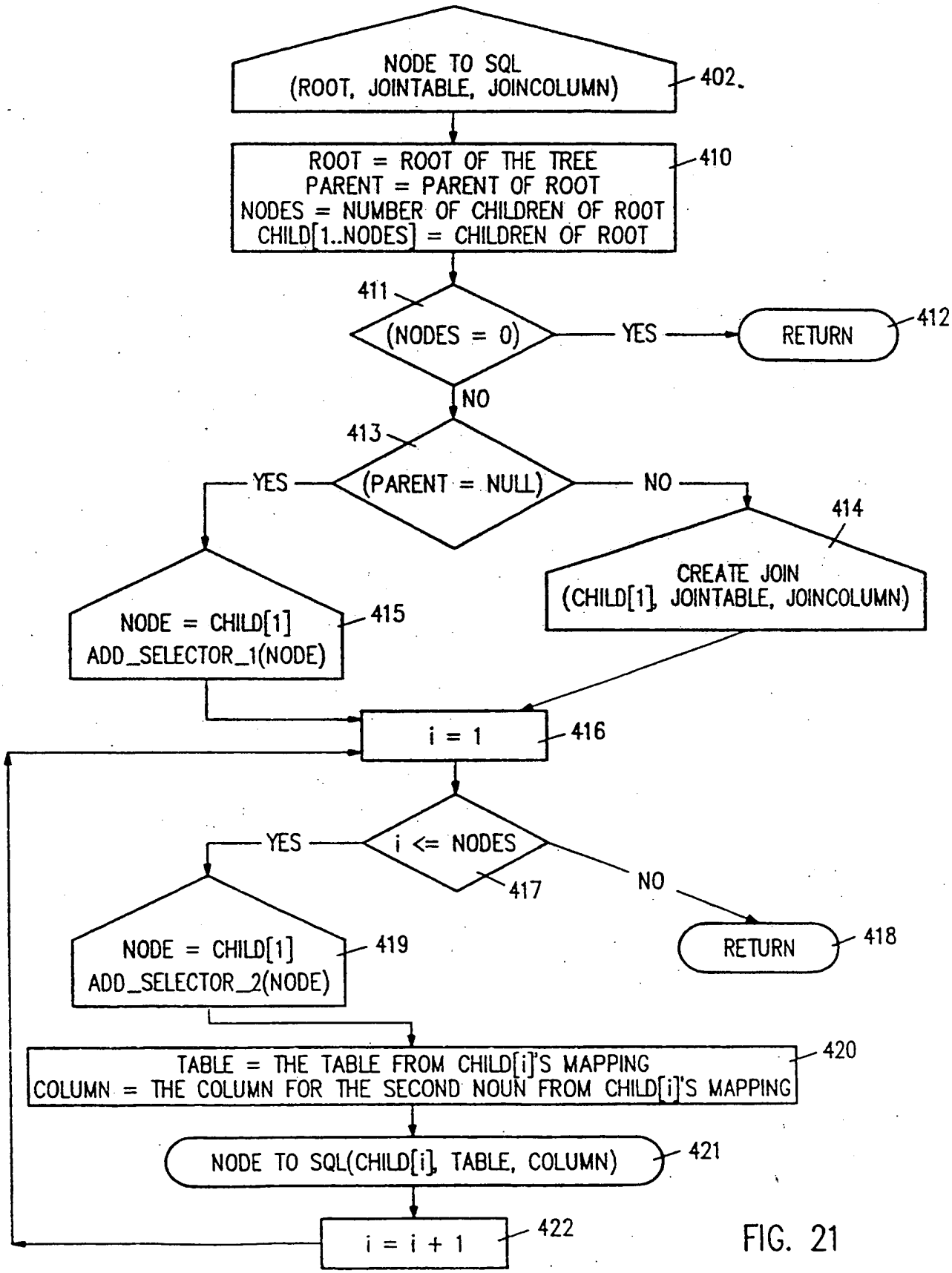

NODE TO SQL
(ROOT, JOINTABLE, JOINCOLUMN)
402
ROOT = ROOT OF THE TREE
PARENT = PARENT OF ROOT
NODES = NUMBER OF CHILDREN OF ROOT
CHILD[1..NODES] = CHILDREN OF ROOT
410
411
(NODES = 0)
YES
RETURN
412
NO
413
(PARENT = NULL)
YES
NO
414
CREATE JOIN
(CHILD[1], JOINTABLE, JOINCOLUMN)
NODE = CHILD[1]
ADD_SELECTOR_1(NODE)
415
i = 1
416
i <= NODES
YES
NO
417
NODE = CHILD[1]
ADD_SELECTOR_2(NODE)
419
RETURN
418
TABLE = THE TABLE FROM CHILD[i]'S MAPPING
COLUMN = THE COLUMN FOR THE SECOND NOUN FROM CHILD[i]'S MAPPING
420
NODE TO SQL(CHILD[i], TABLE, COLUMN)
421
i = i + 1
422

FIG. 21

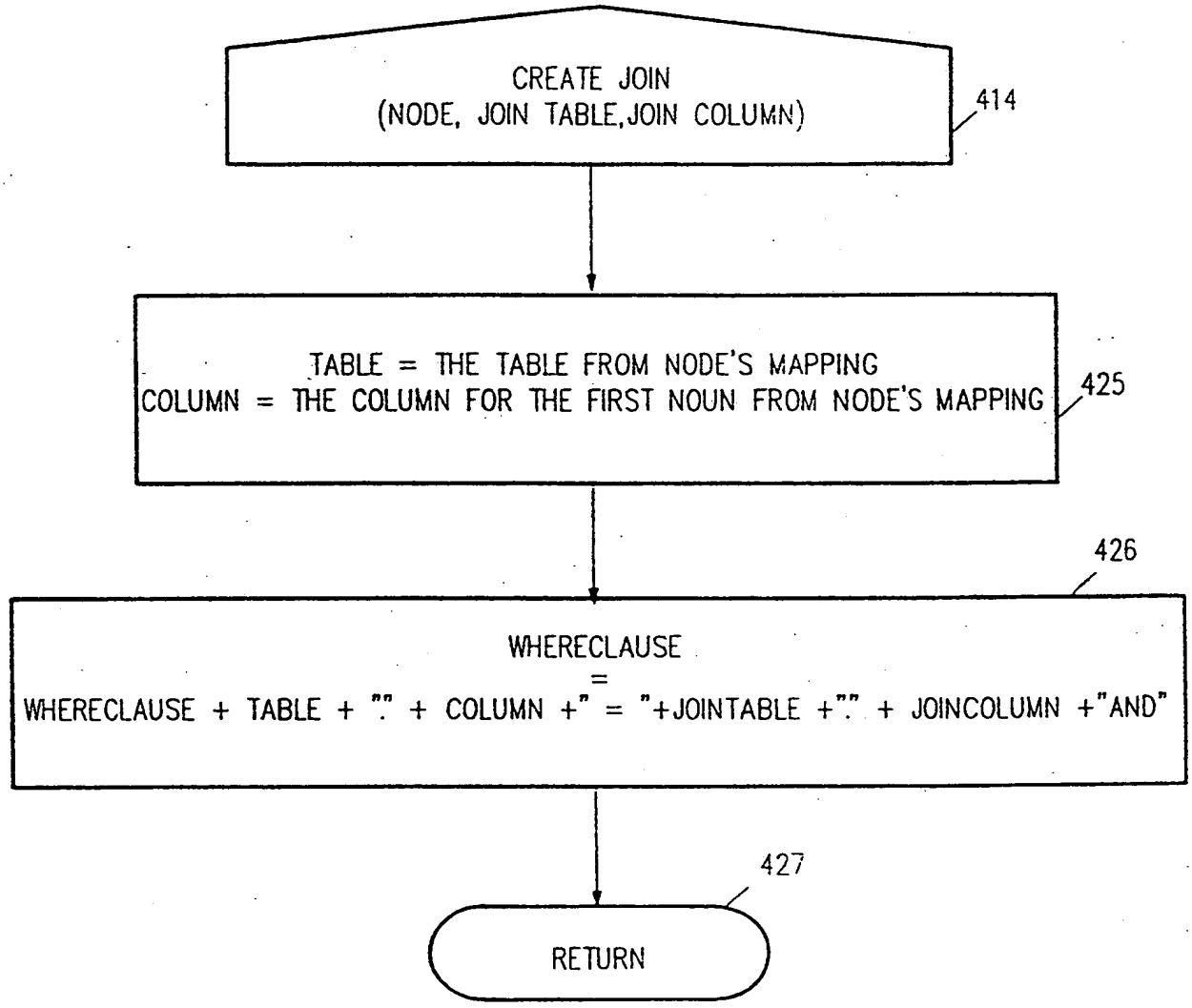

CREATE JOIN
(NODE, JOIN TABLE,JOIN COLUMN)
414
TABLE = THE TABLE FROM NODE'S MAPPING
COLUMN = THE COLUMN FOR THE FIRST NOUN FROM NODE'S MAPPING
425
426
WHERECLAUSE
=
WHERECLAUSE + TABLE + "." + COLUMN +" = "+JOINTABLE +"." + JOINCOLUMN +"AND"
427
RETURN

FIG. 22

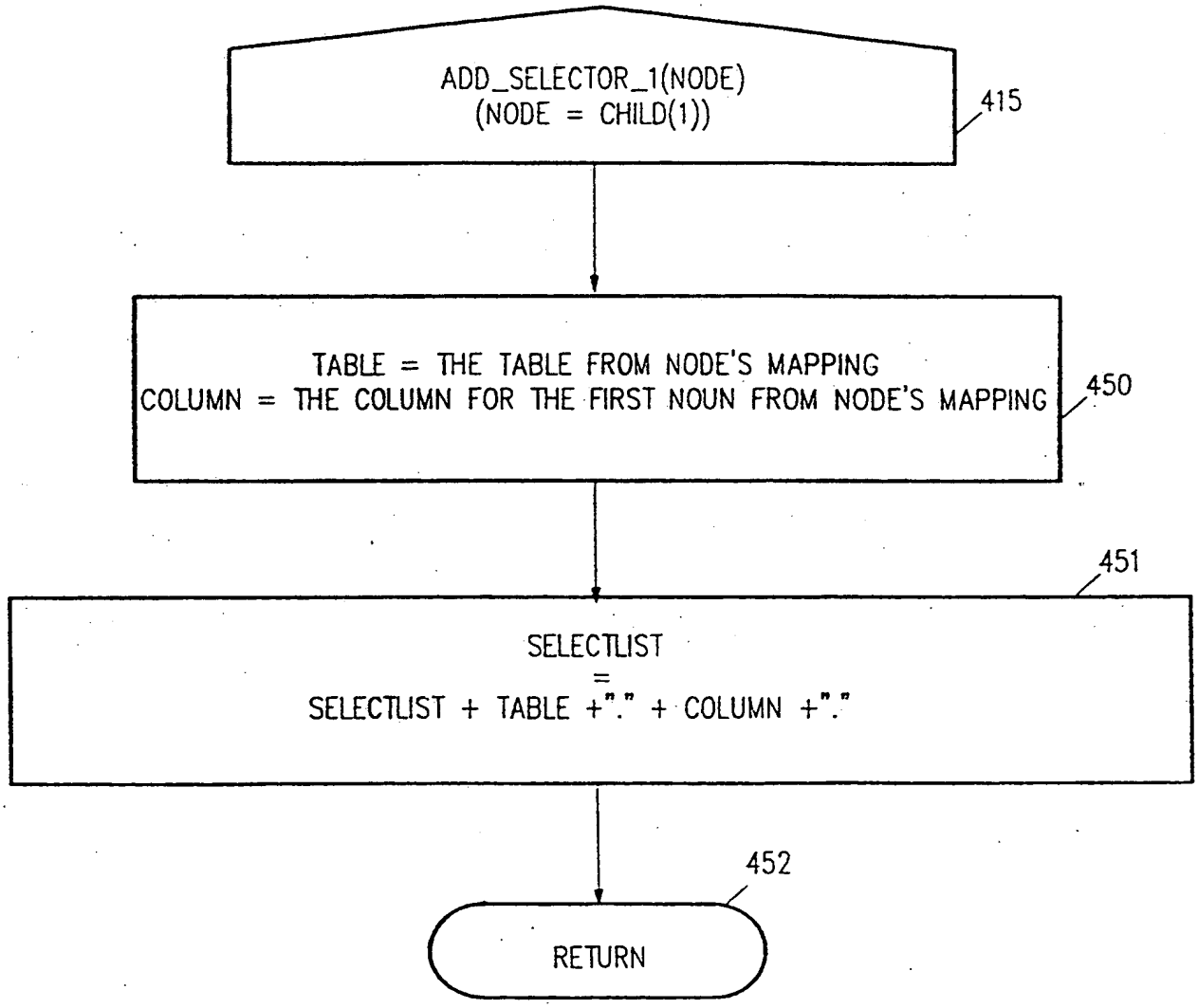

ADD_SELECTOR_1(NODE)
(NODE = CHILD(1))
415
TABLE = THE TABLE FROM NODE'S MAPPING
COLUMN = THE COLUMN FOR THE FIRST NOUN FROM NODE'S MAPPING
450
451
SELECTLIST
=
SELECTLIST + TABLE +"." + COLUMN +"."
452
RETURN

FIG. 23

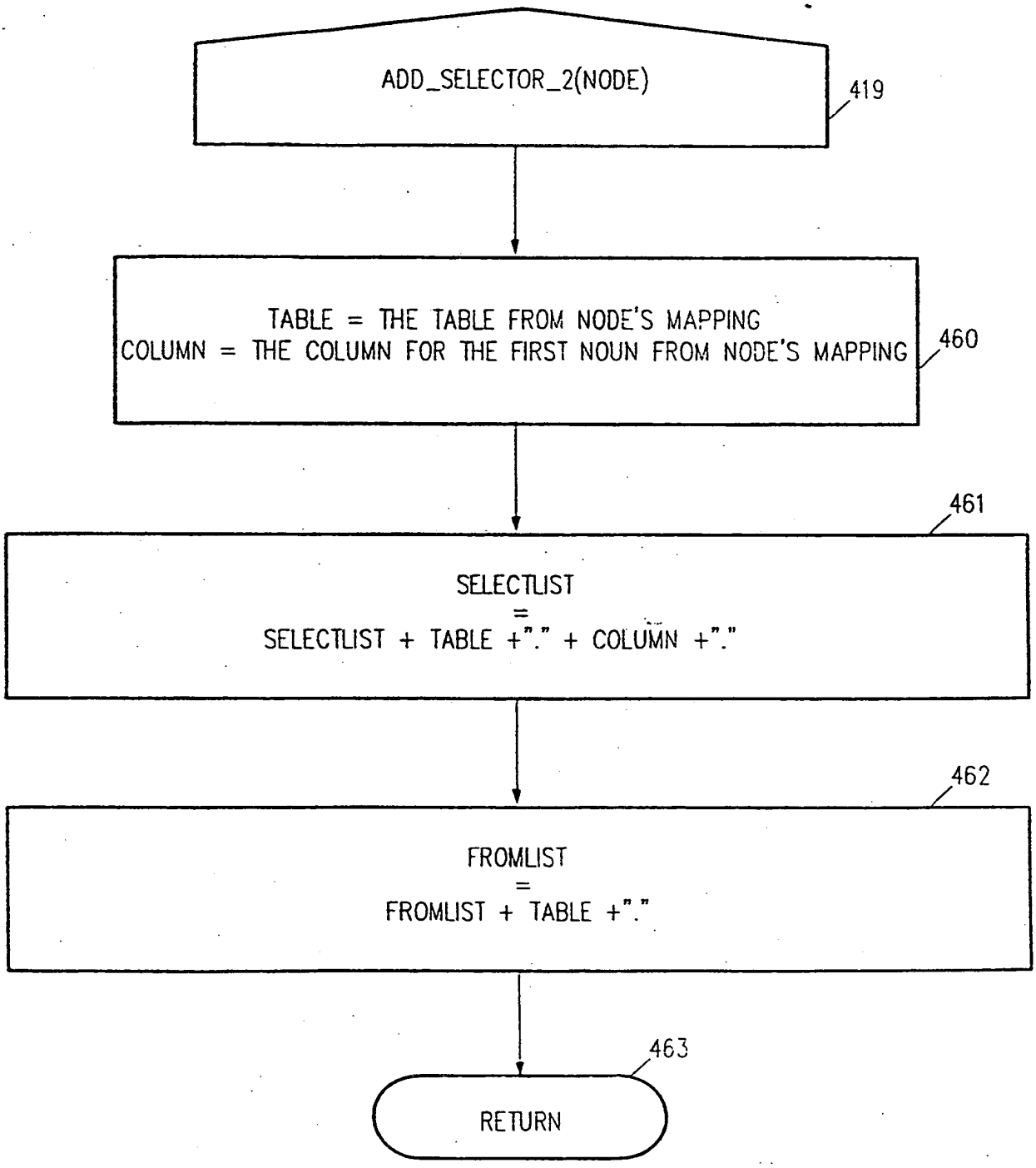

ADD_SELECTOR_2(NODE)
419
TABLE = THE TABLE FROM NODE'S MAPPING
COLUMN = THE COLUMN FOR THE FIRST NOUN FROM NODE'S MAPPING
460
461
SELECTLIST
=
SELECTLIST + TABLE +"." + COLUMN +"."
462
FROMLIST
=
FROMLIST + TABLE +"."
463
RETURN

FIG. 24

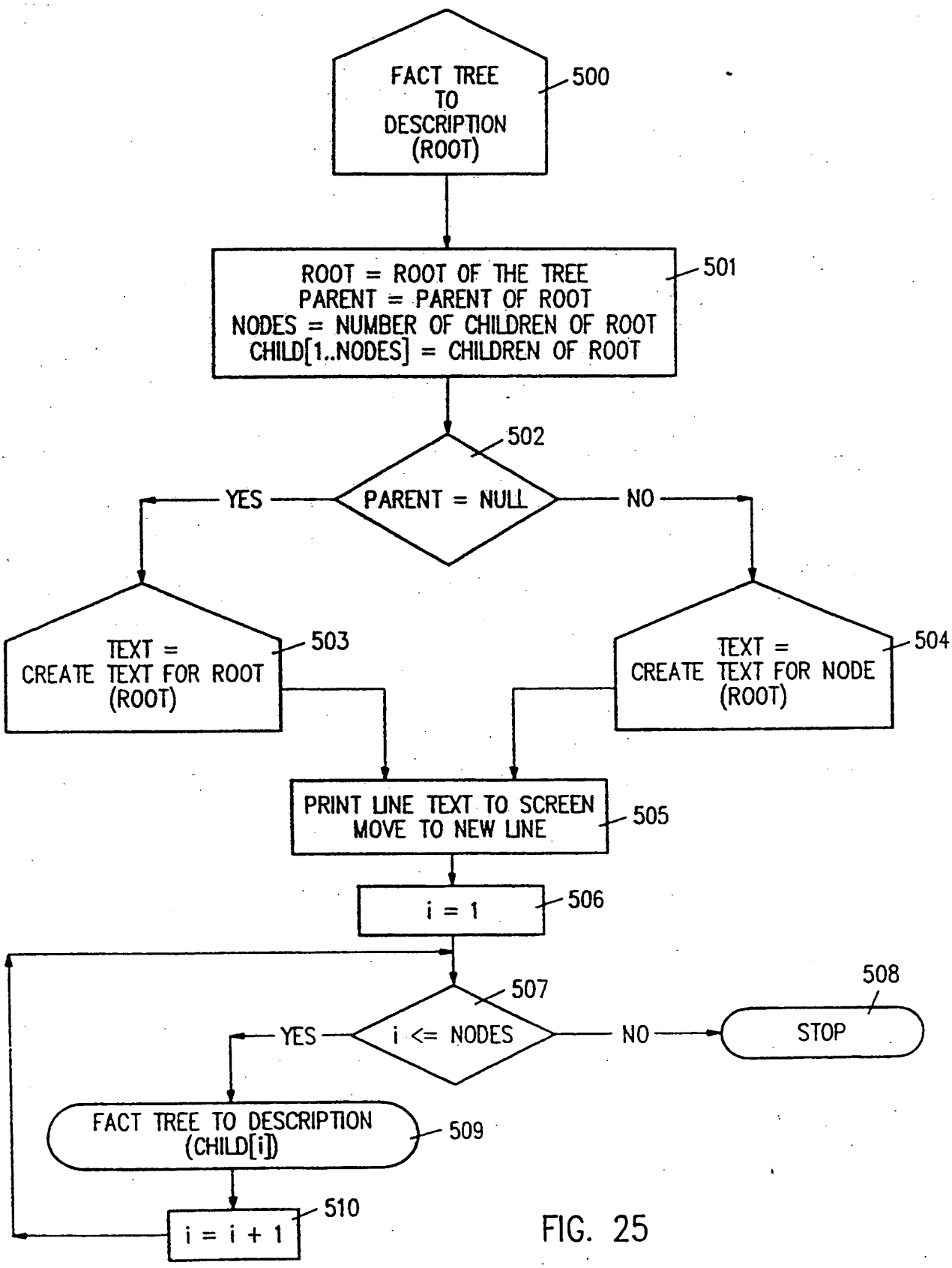

FACT TREE TO DESCRIPTION (ROOT)
500
ROOT = ROOT OF THE TREE
PARENT = PARENT OF ROOT
NODES = NUMBER OF CHILDREN OF ROOT
CHILD[1..NODES] = CHILDREN OF ROOT
501
502
PARENT = NULL
YES
NO
TEXT = CREATE TEXT FOR ROOT (ROOT)
503
TEXT = CREATE TEXT FOR NODE (ROOT)
504
PRINT LINE TEXT TO SCREEN
MOVE TO NEW LINE
505
i = 1
506
507
i <= NODES
YES
NO
508
STOP
FACT TREE TO DESCRIPTION (CHILD[i])
509
i = i + 1
510

FIG. 25

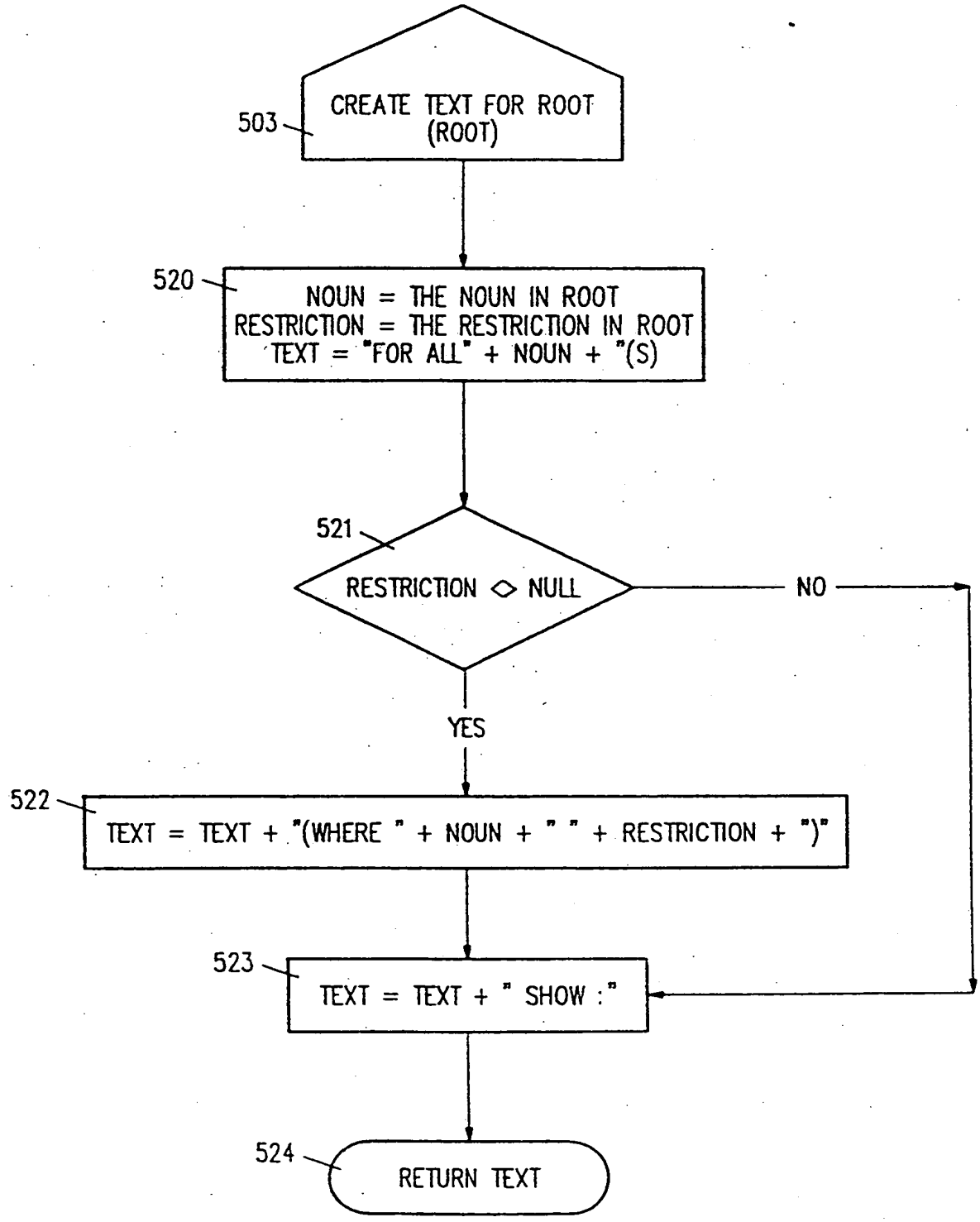

503
CREATE TEXT FOR ROOT
(ROOT)
520
NOUN = THE NOUN IN ROOT
RESTRICTION = THE RESTRICTION IN ROOT
TEXT = "FOR ALL" + NOUN + "(S)
521
RESTRICTION ◇ NULL
NO
YES
522
TEXT = TEXT + "(WHERE " + NOUN + " " + RESTRICTION + ")"
523
TEXT = TEXT + " SHOW :"
524
RETURN TEXT

FIG. 26

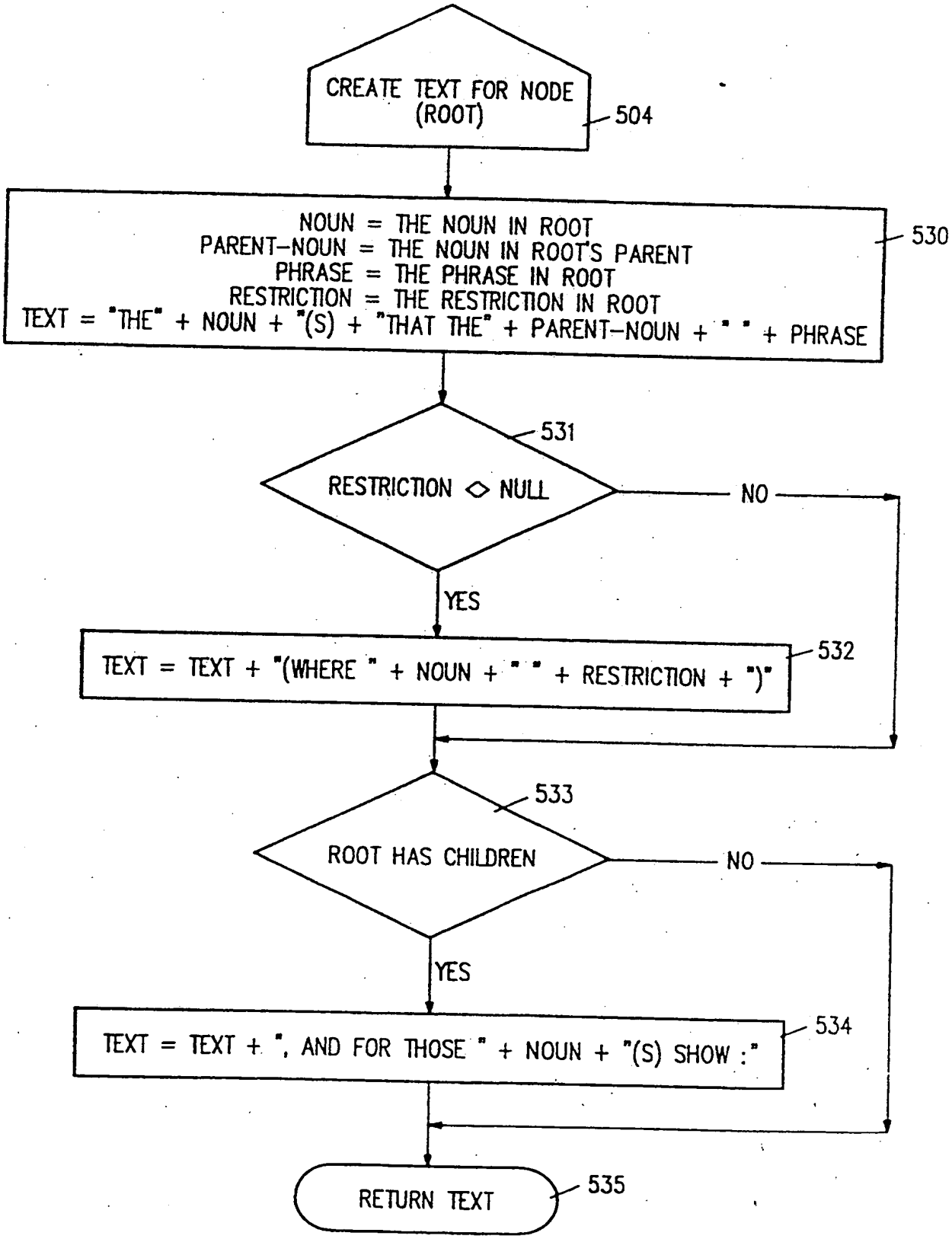

CREATE TEXT FOR NODE (ROOT)
504
NOUN = THE NOUN IN ROOT
PARENT-NOUN = THE NOUN IN ROOT'S PARENT
PHRASE = THE PHRASE IN ROOT
RESTRICTION = THE RESTRICTION IN ROOT
TEXT = "THE" + NOUN + "(S) + "THAT THE" + PARENT-NOUN + " " + PHRASE
530
531
RESTRICTION ◇ NULL
NO
YES
TEXT = TEXT + "(WHERE " + NOUN + " " + RESTRICTION + ")"
532
533
ROOT HAS CHILDREN
NO
YES
TEXT = TEXT + ", AND FOR THOSE " + NOUN + "(S) SHOW :"
534
RETURN TEXT
535

FIG. 27

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/09658

A. CLASSIFICATION OF SUBJECT MATTER
IPC(6) :G06F 17/30, 17/40
US CL :395/600, 155
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 395/600, 155

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
APS (USPAT)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y,P	US, A, 5,088,052 (Spielman et al) 11 February 1994, Figures 1-2, col. 2, line 43 - col. 5, line 12.	1-42
Y,P	US, A, 5,257,365 (Powers et al) 26 October 1993, Figures 2-8, col. 3, line 18 - col. 7, line 9.	43-66
Y,P	US, A, 5,301,313 (Terada et al) 05 April 1994, Figures 2, 12-16, col. 2, line 52 - col. 3, line 26, col. 4, line 45 - col. 9, line 61.	1-42
Y,P	US, A, 5,247,666 (Buckwold) 21 September 1993, Figures 5-15, col. 6, line 28 - col. 10, line 13.	43-66

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search	Date of mailing of the international search report
24 OCTOBER 1994	30 JAN 1995
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	Authorized officer PAUL HARRITY
Facsimile No. NOT APPLICABLE	Telephone No. (703) 305-9677

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/09658

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US, A, 5,175,814 (Anick et al) 29 December 1992, Figures 18-23, col. 14, line 30 - col. 16, line 25, col. 18, line 54 - col. 22, line 39.	43-66
A	US, A, 5,197,005 (Shwartz et al) 23 March 1993, Figures 1-4,7, col. 6, line 19 - col. 16, line 54.	43-66
A	US, A, 4,829,427 (Green) 09 May 1989, Figures 6-8, 12-14, col. 6, line 48 - col. 9, line 57, col. 9, line 60 - col. 10, line 21.	43-66